
시작합니다!

강화학습 & Python



강화학습 시작하기



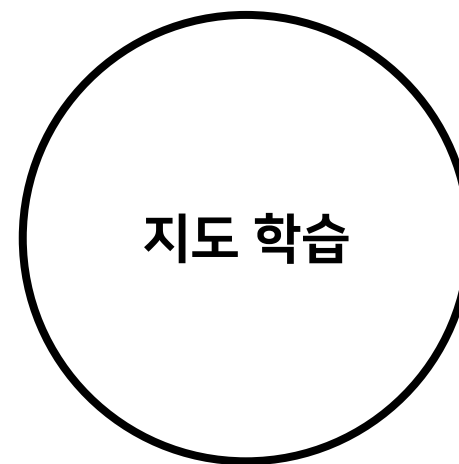
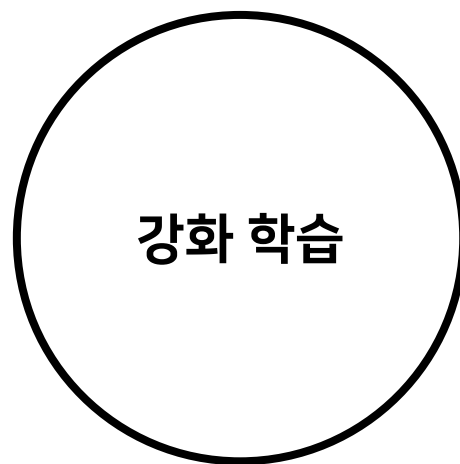


강화 학습에 대해 배우는 긴 여정이 시작되었습니다. 이번 강의의 첫 번째 목표는 강화 학습이란 무엇인지에 대해 설명하는 것입니다. 그에 더해서 강화 학습이 얼마나 직관적이고 매력적인 아이디어인지 살펴보고자 하는 것입니다. 이를 위해 먼저 강화 학습이 대체 무엇인지 지도 학습과 비교하여 설명하며 이야기를 시작하겠습니다. 이어서 강화 학습이 풀고자 하는 문제와 강화 학습에서 자주 등장하는 기초 개념들을 살펴본 후에 마지막으로 동기 부여를 위해 강화 학습이 갖고 있는 위력에 대해 간략히 다루겠습니다.

강화 학습이 어떤 특징을 가지고 있는지 쉽게 알아보기 위해 먼저 어린 아이가 자전거를 배우는 이야기를 잠시 해보겠습니다. 6살 어린이 에이림과 멧쟁이 사자는 자전거를 타는 법을 배우기 위해 둘 다 자전거를 끌고 운동장에 갔습니다. 여기까지는 똑같습니다. 하지만 둘은 서로 다른 방식을 택합니다. 에이림은 자전거 선생님에게 배웁니다. 자전거를 능숙하게 탈 줄 아는 선생님은 강의를 시작합니다. 핸들은 어떻게 잡는지, 페달은 어떻게 밟는지, 방향은 어떻게 바꾸는지 하나하나 알려주었습니다. 강의를 끝나고 나서는 실습시간도 있었습니다. 선생님은 먼저 자전거를 능숙하게 타는 모습을 보여주고, 그 다음은 에이림을 자전거에 앉혀놓고 세세하게 코치해 줬습니다. 말하자면 에이림은 **지도자(supervisor)**의 도움을 받아서 자전거를 타는 법을 **학습(learning)**하고 있습니다.

반면 멧쟁이 사자는 조금 다른 방법으로 도전합니다. 혼자서 씩씩하게 헬멧을 착용하고 혼자서 운동장으로 향합니다. 그리고 크게 심호흡을 한 번 하고는 무작정 자전거에 올라 타서 페달을 밟기 시작합니다. 물론 얼마 못 가 쿵! 하고 넘어졌지만 멧쟁이 사자에게 포기란 없었습니다. 일어나서 흙을 털고는 다시 자전거에 올라타입니다. 계속해서 타고 넘어지고 다시 타는 것을 반복합니다. 여기서 중요한 점은 아무도 이런 지식을 알려준 적이 없다는 것입니다. 그럼에도 불구하고 스스로 조금씩 깨우치게 되는 것이죠. 그런 깨우침을 통해 균형을 잡고 버티는 시간도 조금씩 늘어나게 됩니다. 말하자면 **시행착오(trial and error)**를 통해 학습하고 있습니다.

둘 다 자전거를 잘 타는 것을 목표로 하고 있지만, 전혀 다른 접근 방법을 통해 학습 중입니다. 에이림은 지도자의 도움을 통해 자전거 타는 법을 배웠고, 멧쟁이 사자는 스스로 시행착오를 통해 배우게 됩니다. 이 2가지 카테고리의 방법론은 꼭 자전거를 배울 때에만 해당하는 이야기는 아닙니다. 두 방법론 모두 일반적으로 무언가에 대해 학습할 때에 취할 수 있는 대표적인 방법론입니다. 심지어 기계가 무엇인가를 학습할 때에도 위 2가지 방법론 중 하나를 채택할 수 있습니다. 방법론 중 에이림의 방법은 **지도 학습(supervised learning)**에 대응되고, 멧쟁이 사자의 방법은 우리의 주제인 **강화 학습(reinforcement learning)**에 대응됩니다.



멋쟁이 사자는 각 상황에서 어떻게 해야 하는지 정답을 알려줄 지도자가 없었습니다. 가르쳐 주는 사람이 없기 때문에 모든 지식을 혼자서 터득해야 했습니다. 그러면 어떻게 자전거를 탈 수 있는 지식을 깨우칠 수 있었을까요?

멋쟁이 사자가 자전거를 배우는 과정은 강화학습과 꼭 닮아 있습니다. 여기서 강화 학습이 무엇인지 2가지 설명으로 소개하겠습니다.

▶ 쉽지만 추상적인 설명

‘시행착오를 통해 발전해 나가는 과정’

▶ 어렵지만 좀 더 정확한 설명

*‘순차적 의사결정 문제에서 누적 보상을 최대화 하기 위해
시행착오를 통해 행동을 교정하는 학습 과정’*

비록 어렵지만 정확한 버전의 문장이 지금은 낯선 단어가 포함되어 있다 보니 어렵게 느껴지겠지만 천천히 하나씩 배워 나가보도록 하겠습니다.

이제부터 각 개념을 차례로 짚어가며 본격적으로 강화 학습에 대한 이야기를 시작해 보겠습니다.

그 시작은 ‘순차적 의사결정 문제’입니다.

강화 학습이 풀고자 하는 문제는 바로 **순차적 의사결정(Sequential decision making)** 문제입니다. 순차적 의사결정 문제라는 말이 낯설게 들리지만 사실 우리 삶 곳곳에서 맞닥뜨릴 수 있습니다. 예컨대 샤워를 하는 상황을 살펴보죠.

우리는 샤워를 할 때 크게 4단계를 거칩니다.

① 옷을 벗는다. ② 샤워를 한다. ③ 물기를 닦는다. ④ 옷을 입는다.

그리고 이 4단계는 반드시 순서에 맞게 이루어져야 합니다. 만일 순서가 뒤바뀌면 어떻게 될까요?

→ ④ 옷을 입고, ② 샤워를 하고, ③ 물기를 닦고, ① 옷을 벗는다.

→ ③ 물기를 닦고, ② 샤워를 하고, ① 옷을 벗고, ④ 옷을 입는다.

이처럼 아무리 간단한 과정이라 하더라도 이를 성공적으로 마치기 위해서 우리는 몇 가지 의사결정을 순차적으로 해 주어야 합니다. 어떤 행동(의사결정)을 하고, 그로 인해 상황이 바뀌고, 다음 상황에서 또 다시 어떤 행동을 하고... 이처럼 각 상황에 따라 하는 행동이 다음 상황에 영향을 주며, 결국 연이은 행동을 잘 선택해야 하는 문제가 바로 **순차적 의사결정 문제**입니다.

순차적 의사결정 문제에서

세상에는 순차적 의사결정 문제가 정말 많고 다양합니다. 왜냐하면 이 복잡한 세상에 어떤 행동을 하고 단순하게 성공/실패라는 결과가 나오고 바로 끝나버리는 경우는 많지 않기 때문입니다. 대부분 행동을 하면 상황이 바뀌고, 거기서 또 다른 적절한 행동을 취해야 합니다. 예컨대 다음과 같습니다.

▶ 주식 투자에서의 포트폴리오 관리



주식 투자는 정해진 예산으로 어떤 주식을 어느정도 들고 있을지에 대한 선택의 연속입니다. 해당 주식을 사는 순간부터 시작되어, 그 주식을 계속 가지고 있을지, 팔지, 팔고 나서 어떤 다른 주식을 살지 계속해서 결정해야 합니다. 매 순간 내가 어떤 주식을 어느정도 가지고 있는지에 따라 얻게 되는 수익은 천차만별로 바뀝니다. 게다가 그에 따라 이후의 전략도 달라집니다. 결국 시장 상황에 따라 리스크는 줄이고 기대 수익률은 올릴 수 있는 방향으로 유연하게 포트폴리오를 재구성해야 합니다. 따라서 주식 투자에서의 **포트폴리오 관리**는 **순차적 의사결정 문제**로 볼 수 있습니다.

▶ 운전

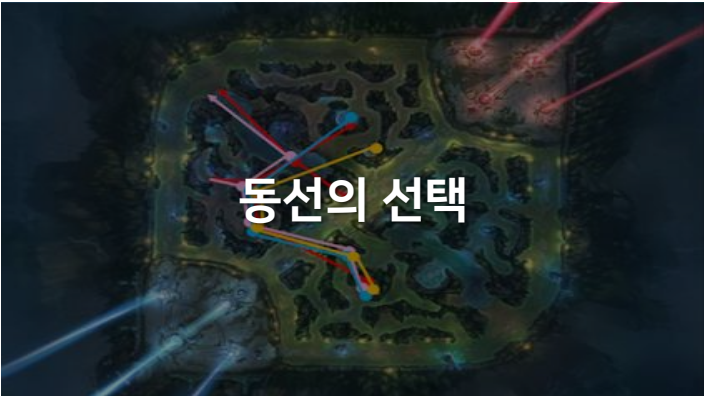
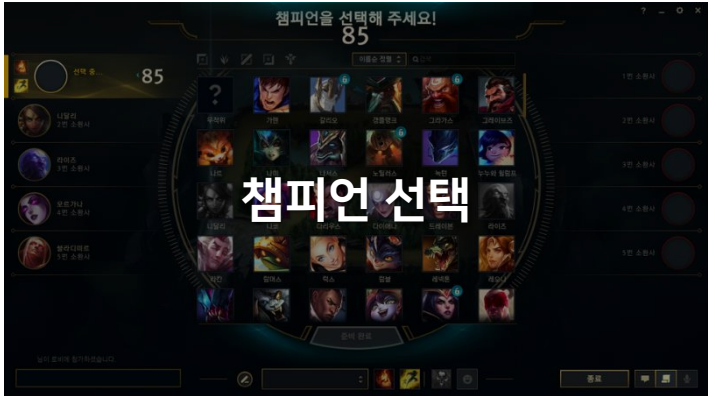
자전거와 마찬가지로 운전 또한 순차적 의사결정 문제의 관점에서 생각해 볼 수 있습니다. 예를 들어 서울에서 강원도 횡성까지 2시간 동안 운전을 해서 간다고 할 때 매 순간 내리는 모든 결정은 이후의 결정에 영향을 줍니다. 먼저 큰 스케일에서는 어느 도로를 이용할지, 고속도로를 탈지 국도를 탈지 등을 결정해야 합니다. 그때 선택을 잘 해서 막히지 않는 도로를 탔다면 이후의 여정은 순탄할 것입니다.

그만큼 1번의 의사결정이 이후 2시간 동안의 운전 양상을 결정하게 되는 것입니다. 보다 작은 스케일에서는 순간마다 어느 차선을 이용할지 선택해야 합니다. 한 번 선택을 하면 다양한 시간 스케일에서 연쇄 작용처럼 영향을 줍니다. 이처럼 운전하는 상황은 큰 틀에서도 작은 틀에서도 모두 순차적 의사결정 과정입니다.



▶ 게임

대부분의 게임은 순차적 의사결정의 정수라고 할 수 있을 만큼 모든 의사결정이 시간에 따라 순차적으로 이루어지며, 그만큼 빠르고 정확한 의사결정을 필요로 합니다. 대표적으로 리그 오브 레전드(LOL)와 같은 게임에서는 처음에 어떤 챔피언을 선택하여 플레이 할지 정하는 순간부터 의사결정이 시작됩니다. 영웅을 정하고 나면 이후 게임에서 겪는 경험이 통째로 바뀌게 됩니다. 바둑이나 체스와 같은 보드 게임도 마찬가지 입니다. 대부분의 게임은 **잘게 쪼개진 선택의 연속**으로 이루어져 있으며, 각 선택에 따라 이후 겪는 상황이 변하기 때문에 주어진 상황 안에서 최선의 선택을 내려야 합니다.



보다시피 세상에는 순차적 의사 결정 문제가 아닌 것을 찾기가 더 어려울 정도로 순차적 의사결정 문제는 많은 상황에서 발생합니다. 그만큼 중요한 문제 입니다. 그리고 강화 학습은 이런 중요한 문제를 풀기 위해 고안된 방법론입니다. 그렇기 때문에 자연스레 강화 학습이 인공지능에서 얼마나 중요한 비중을 차지할지 감이 올 것입니다. 지금까지 앞서 어려운 설명의 정의에 등장했던 **순차적 의사결정 문제라는 개념**에 대해 설명했습니다. 이제 **보상**이라는 개념으로 넘어가 보겠습니다.

보상(reward)이란 의사결정을 얼마나 잘하고 있는지 알려주는 신호입니다. 그리고 강화 학습의 목적은 과정에서 받는 보상의 총합, 즉 **누적 보상(cumulative reward)**을 최대화 하는 것입니다. 앞서 멧쟁이 사자가 자전거를 타는 것을 혼자서도 학습할 수 있었던 이유도 사실 보상이라는 신호가 있었던 덕분입니다. 이 경우에 보상은 자전거를 잘 타고 있는지를 알려주는 시그널입니다. 잘 타고 있다면 보상이 크고, 못 타고 있다면 보상이 작을 것입니다. 그렇다면 멧쟁이 사자에게 보상은 무엇일까요?

예컨대 안 넘어지고 1m를 갈 때마다 +1이라는 식으로 보상을 정해볼 수 있습니다. 정확히 멧쟁이 사자의 뇌 속에서 미터기를 가지고 측정하여 1m마다 1점을 주지는 않았겠지만, 대충 이와 비슷한 기준을 통해 뇌 안에서 잘 하고 있는지를 판별하고 있었을 것입니다. 1m당 1점이 보상이라면 자연스레 안 넘어지고 **최대한 나아가는 것이 학습의 목적**이 됩니다. 보상은 강화 학습을 이해하는데 있어서 가장 중요한 개념입니다. 보상이 없다면 그 어떤 것도 배울 수 없습니다. 이렇게 중요한만큼 **보상을 3가지 특징**으로 나누어 자세히 설명해보겠습니다.



▶ 어떻게 X 얼마나 O

보상의 첫 번째 특징은 보상은 '어떻게'에 대한 정보를 담고 있지 않다는 점입니다. 보상은 내가 어떤 행동을 하면 그것에 대해 '얼마나' 잘 하고 있는지 평가를 해줄 뿐, 어떻게 해야 높은 보상을 얻을 수 있을지 알려주지 않습니다. 그런 의미에서 보상은 지도 학습의 정답과 질적으로 다릅니다. 그러면 '어떻게'에 대한 정보를 아무도 알려주지 않는데 어떻게 학습을 할 수 있는 것일까요?
그것은 바로 수많은 시행착오 덕분입니다.

멋쟁이 사자는 운동장에서 200번도 더 넘어지면서 계속해서 이런 저런 시행착오를 겪고 있습니다. 페달을 세게도 밟아보고, 약하게도 밟아보고, 핸들을 왼쪽으로도 꺾어보는 등 이런저런 시도를 하다보면 "어라?" 하면서 본인도 모르게 잘 타지는 경우가 있습니다. 매번 1m도 못 가서 넘어지다가 스무 번째 시도에서 처음으로 3m를 가게 된 것입니다. 평소보다 더 먼 거리를 가는데 우연히 성공했고 비로소 깨닫게 됩니다.
"방금 내가 한 행동은 이전에 1m도 못 가서 넘어질 때 했던 행동들보다 좋은 행동이었구나." 하고 말합니다. 그래서 멋쟁이 사자는 방금했던 행동들을 옳은 행동이라 생각하고 더 자주 하도록 교정합니다.

반대의 이야기도 충분히 가능합니다. 멋쟁이 사자는 이제 3m도 어렵지 않게 갈 수 있게 되었습니다. 보다 멀리 가고 싶어서 또 이런 저런 시도를 해볼 것입니다. 그러다가 이번에는 페달을 평소보다 더 약하게 밟는 시도를 해 보았고 결과적으로 3m보다 짧은 거리에서 넘어졌습니다. 그러면 멋쟁이 사자는 다음과 같은 생각을 하게 됩니다. "페달을 너무 약하게 밟으면 쉽게 넘어지는구나. 다음 번엔 조금 더 세게 밟아야겠다."

CHAPTER 0 – 강화학습 시작하기

멋쟁이 사자가 자신의 행동을 교정할 수 있는 이유는 잘 했을 때는 잘했다하고, 못 했을 때는 못 했다고 평가해 주는 **신호**가 있기 때문입니다. 그것이 바로 **보상**입니다. 보상이 어떻게 해야 할지를 직접적으로 알려주지는 않지만, 사후적으로 보상이 낮았던 행동들은 덜 하고, 보상이 높았던 행동들은 더 하면서 **보상을 최대화 하도록 행동을 조금씩 수정해 나갑니다.**



▶ 스칼라

보상의 두 번째 특징은 보상이 **스칼라(scalar)**라는 점입니다. 스칼라는 벡터(vector)와는 다르게 크기를 나타내는 값 하나로 이루어져 있습니다. 보상이 벡터라면 동시에 2개 이상의 값을 목표로 할 수 있겠지만 스칼라이기 때문에 오직 하나의 목적만을 가져야 합니다.

여기서 한 번 시간을 갖고 생각해볼길 바랍니다. 이는 자연스러운 가정일까요?

예를 들어 어떤 대학교의 신입생은 대학 생활 동안 여러 목표를 가질 수 있습니다. 학점도 잘 받고 싶고, 동아리 활동도 재미있게 하고 싶고, 또 연애도 하고 싶습니다. 무엇 하나 포기할 수 없습니다. 여기서 학점을 x , 동아리 활동을 y , 연애를 z 라고 하면 결국 이 학생의 목표는 (x, y, z) 라는 벡터로 표현할 수 있습니다. 학점을 잘 받을수록, 동아리에 들어가 즐거운 시간을 보낼수록, 좋은 사람을 만나 행복한 연애를 할수록 x, y, z 각각의 값이 증가합니다. 하지만 **강화학습에서의 보상은 스칼라**여야 한다는 점이 발목을 잡습니다. 때문에 이 목표들을 잘 섞어서 하나의 스칼라 형태로 표현해줘야 합니다.

어떻게 하나의 스칼라로 표현할 수 있을까요?

여러 방법이 있겠지만 그 중 하나는 바로 **가중치를 두는 방법**입니다. 학점을 중요시 하는 학생은 각각 (50%, 25%, 25%)의 가중치를 뒤서 $0.5x + 0.25y + 0.25z$ 라는 식으로 표현할 수 있습니다. 이렇게 표현하면 3개의 목표가 하나의 숫자로 합쳐져서 스칼라가 됩니다.

CHAPTER 0 – 강화학습 시작하기

강화 학습은 스칼라 형태의 보상이 있는 경우에만 적용할 수 있습니다. 이런 설정에는 충분히 의문을 제기해볼 수 있습니다. 세상은 매우 복잡하고 단 하나의 목표만 존재하는 단순한 상황이 많지 않으니까요. 만일 어떤 문제는 도저히 하나의 목표만을 설정하기 어렵다면 그 문제에 강화 학습을 적용하는 것은 적절하지 못할 수 있습니다. 혹은 문제를 단순화하여 하나의 목표를 설정해야 합니다. 대신 잘 정해진 하나의 숫자로 된 목표가 있다면, 강화 학습은 불도저처럼 해당 목표를 최대한으로 취하도록 최적화할 것입니다.

스칼라 보상의 예시로는 다음과 같습니다.



하나의 문제 안에서도 다양한 방법으로 보상을 설정할 수 있습니다. 자전거 예시를 살펴보면 앞서 멧쟁이 사자의 목적은 결국 안 넘어지고 자전거를 타는 것이었고 보상은 1m 갈 때마다 +1이라고 정했었습니다. 그러면 3m를 안 넘어지고 탔을 때는 3점, 2m 탔을 때는 2점이니, 3m 탔던 경우가 더 잘한 것임을 알 수 있습니다. 누적 보상을 최적화 한다는 것은 결국 안 넘어지고 갈 수 있는 거리를 최대한으로 늘리겠다는 것이니 자전거를 배우는 목적에 부합합니다.

CHAPTER 0 – 강화학습 시작하기

다른 방법으로는 **시간의 관점**에서 1초 안 넘어지고 탈 때마다 +1이라고 정할수도 있습니다. 10초를 타면 10점, 5초를 타면 5점이니 10초를 탄 쪽이 더 잘 탄것입니다. 이처럼 주어진 목적을 이루기 위해 **그와 연관된 보상을 정할 수 있는 방법은 다양**하며 목적과 누적 보상의 값이 의미하는 바가 잘 대응되도록 정해야 합니다.



▶ 희소하고 지연된 보상

보상의 세 번째 특징은 보상이 **희소(sparse)**할 수 있으며 또 **지연(delay)**될 수 있다는 점입니다. 행동과 보상이 일대일로 대응이 된다면 강화 학습은 한결 쉬워집니다. 행동에 대한 평가가 즉각적으로 이루어지는 만큼 좋은 행동과 안 좋은 행동을 가려내기 쉽기 때문입니다. 하지만 **보상은 선택했던 행동의 빈도에 비해 훨씬 가끔 주어지거나, 행동이 발생한 후 한참 뒤에 나올 수 있고 이 때문에 행동과 보상의 연결이 어려워 집니다.** 만일 행동마다 보상이 있는 것이 아니라 행동을 10번 해야 보상이 1번 나타나거나, 행동을 10번 하고 나서야 과거 처음 한 행동의 보상이 주어진다면 **이 보상이 어떤 행동 덕분인지 책임 소재가 불분명해지면서 그만큼 학습도 어려워 집니다.**

바둑의 경우를 생각해 보겠습니다. 바둑의 목적은 경기를 이기는 것이기 때문에 이기면 +1, 패배하면 -1의 형태로 보상을 정의할 수 있습니다. 그런데 평균적으로 250수는 뒤야 한 경기가 끝이 납니다. 그러면 **행동 250개와 보상 1개가 연결이 되는 극한의 상황**에 처하게 됩니다.



250개의 행동 중에는 좋았던 행동도 있고, 안 좋았던 행동도 있을 텐데 어떤 행동을 더 하도록, 어떤 행동을 덜 하도록 어떻게 교정을 할 수 있을까요?

잘은 모르겠지만 “정말 학습이 어렵겠구나” 하고 느껴집니다. 즉 **보상이 희소할수록 학습이 어려워지고** 이런 문제를 해결하기 위해 최근의 강화 학습 연구에서도 **벨류 네트워크(value network)** 등의 다양한 아이디어가 등장하였습니다. 이에 대해서는 천천히 살펴보도록 하겠습니다.

CHAPTER 0 – 강화학습 시작하기

이러한 특성은 지도 학습과 강화 학습을 본질적으로 다르게 만드는 요소이기도 합니다. 지도 학습에서는 정답이 '뒤늦게' 주어지는 경우는 없습니다. 다만 데이터와 정답의 쌍이 있을 뿐이죠. 반면 강화학습에서 다루는 문제는 순차적 의사결정 문제이기 때문에 순차성, 즉 시간에 따른 흐름이 중요하고 이 흐름에서 보상이 뒤늦게 주어지는 것이 가능합니다.

지금까지 강화 학습에서의 보상이라는 개념에 대해서 살펴보았습니다. 또 자전거를 배우는 사례에서 멧쟁이 사자에게 보상은 무엇이며 보상을 통해 행동을 교정하는 법, 나아가 자전거를 잘 타게 되는 과정까지 간략하게 살펴 보았습니다. 강화 학습을 통해 에이전트(agent)가 어떻게 행동을 교정하며 그 목적을 달성하는지는 천천히 배워나가도록 하겠습니다. 그 전에 강화 학습이 풀고자 하는 문제의 상황을 구체화 해보겠습니다.



CHAPTER 0 – 강화학습 시작하기

앞서 설명한 순차적 의사결정 문제를 도식화하면 아래의 그림과 같습니다. 이 그림은 강화 학습을 설명하는데 빠지지 않고 등장하는 도식입니다. 그만큼 강화 학습의 개념을 잘 설명해 줍니다. 에이전트가 **행동(action)**을 하고 그에 따라 상황이 변하는 것을 하나의 **루프(loop)**라 했을 때 이 루프가 끊임없이 반복되는 것을 **순차적 의사결정 문제**라 할 수 있습니다. 이 그림에서 **에이전트(Agent)**와 **환경(Environment)**이라는 개념이 처음 등장한 만큼 그에 대해 먼저 자세히 살펴보겠습니다.



에이전트는 강화 학습의 주인공이자 주체입니다. 학습하는 대상이며 동시에 환경 속에서 행동하는 개체를 가리키는 용어입니다. 자전거 타는 멋쟁이 사자의 예시에서 **멋쟁이 사자가 곧 에이전트** 입니다. 마찬가지로 운전하는 환경에서는 운전자가, 게임을 플레이하는 환경에서는 게임의 플레이어가 에이전트입니다. **에이전트는 어떤 액션을 할지 정하는 것이 가장 주된 역할**입니다. 에이전트 입장에서 위의 루프는 구체적으로 다음 3개의 단계로 이루어져 있습니다.

- 현재 상황 S_t 에서 어떤 액션을 해야 할지 a_t 를 결정
- 결정된 행동 a_t 를 환경으로 보냄
- 환경으로부터 그에 따른 보상과 다음 상태의 정보를 받음

그리고 에이전트를 제외한 모든 요소를 환경이라고 합니다. 그래서 “이것이 환경입니다.” 하고 딱 집어서 정의하기 어렵습니다. 자전거 타는 멧쟁이 사자의 상황에서는 멧쟁이 사자를 제외한 모든 것이 환경입니다. 다른 말로는 멧쟁이 사자가 어떤 행동을 했을 때, 그 결과에 영향을 아주 조금이라도 미치는 모든 요소들이 환경이라고 할 수 있습니다.

멧쟁이 사자가 타고 있는 자전거, 불고 있는 바람, 자전거가 딛고 있는 바닥 등등 모든 것이 환경이 될 수 있습니다. 환경 속에서 에이전트가 어떤 행동을 하고 나면 에이전트의 상태가 바뀝니다. 자전거는 조금 앞으로 나갔을 것이고, 자전거의 기울기나 핸들의 각도도 약간은 변했을 것입니다. 현재 상태에 대한 모든 정보를 숫자로 표현하여 기록해 놓으면 그것을 상태(state)라고 합니다. 현재 자전거의 위치, 기울어진 정도, 핸들의 각도 등등을 숫자로 엮은 하나의 벡터라고 생각하셔도 좋습니다. 그리고 환경은 결국 상태 변화(state transition)를 일으키는 역할을 담당합니다. 행동의 결과를 알려주는 것입니다. 환경이 하는 일은 아래와 같이 단계별로 시작합니다.

1. 에이전트로 부터 받은 액션 a_t 를 통해서 상태 변화를 일으킴
2. 그 결과 상태는 $S_t \rightarrow S_{t+1}$ 로 바뀜
3. 에이전트에게 줄 보상 r_{t+1} 도 함께 계산
4. S_{t+1} 과 r_{t+1} 을 에이전트에게 전달

이처럼 에이전트가 S_t 에서 a_t 를 시행하고, 이를 통해 환경이 S_{t+1} 로 바뀌면, 즉 에이전트와 환경이 한 번 상호 작용하면 하나의 루프가 끝납니다. 이를 한 틱(tick)이 지났다고 표현합니다. 실제 세계는 앞의 그림과 다르게 시간의 흐름이 연속적(continuous)이겠지만, 순차적 의사결정 문제에서는 시간의 흐름을 이산적(discrete)으로 생각합니다.

CHAPTER 0 – 강화학습 시작하기

이산적이란, 뚝뚝 끊어져서 변화가 발생한다는 뜻입니다. 그리고 그 시간의 단위를 틱 혹은 **타임 스텝(time step)**이라고 합니다.

이 개념은 천천히 하나씩 다뤄보도록 할 예정이니 지금 완전히 이해가 가지 않아도 괜찮습니다.



CHAPTER 0 – 강화학습 시작하기

지금까지 강화 학습의 기본 개념들을 살펴 보았습니다. 아마 강화 학습이 어떤 것인지 조금씩 그려질 것입니다. 이번에는 강화 학습의 2가지 매력에 대하여 살펴보겠습니다.

▶ 병렬성의 힘

지금까지 강화 학습을 통해 자전거를 배우는 멧쟁이 사자의 얘기를 했습니다. 그런데 만일 이런 것이 가능하다면 어떨까요? 마치 스타크래프트의 프로토스 처럼 **생각 공유하는 100명의 멧쟁이 사자가 동시에 자전거를 배우는 겁니다.** 자기 자신을 복제해서 멧사1, 멧사2, ..., 멧사100 까지 100명의 멧쟁이 사자가 운동장에 넓게 퍼져서 각자의 자전거를 타고 넘어지기 시작하는 것이죠. 그런데 이 100명은 뇌가 연결되어 있어서 각자가 따로 배우는 것이 아니라 **각자의 경험을 공유**할 수 있습니다.



에이림은 선생님의 도움을 받았기 때문에 2시간 만에 자전거를 배울 수 있었지만 멧쟁이 사자는 혼자서 꼬박 5시간이 필요했습니다. 하지만 멧쟁이 사자는 더 이상 혼자가 아닙니다. 100명의 멧사가 서로 다른 경험을 쌓지만 그 지식이 한데 모여 멧쟁이 사자1이 했던 실수를 나머지 99명은 더 이상 하지 않게 되고, 또 멧쟁이 사자2가 배운 지혜는 나머지 99명이 공유하게 됩니다. 그래서 100명의 멧쟁이 사자는 5시간의 100분의 1에 해당하는 시간인 3분만에 자전거를 배울 수 있습니다. 운동장에 도착해서 채 땀이 나기도 전에 자전거를 마스터 해버린 것입니다.

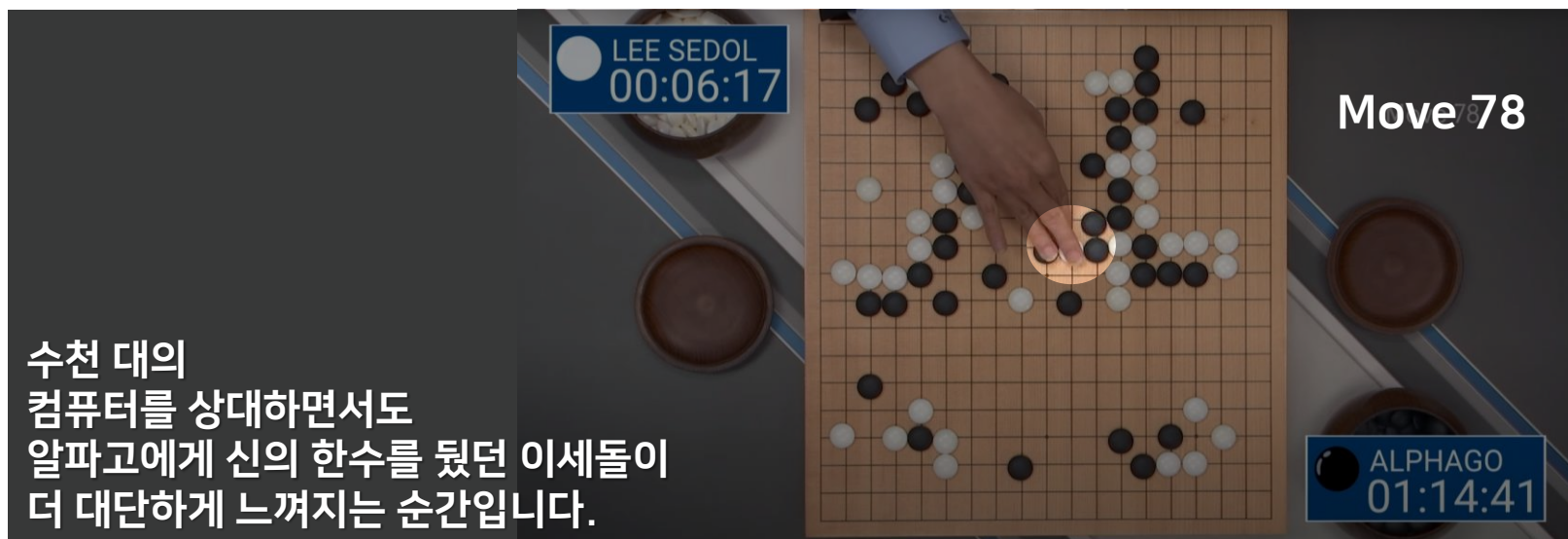
멧쟁이 사자가 실제로 자신의 몸을 100개로 나눌 수는 없겠지만, 이 대상이 사람이 아닌 기계의 경우라면 이야기는 달라집니다.

강화 학습은 **경험을 쌓는 부분의 병렬성을 쉽게 증가**시킬 수 있습니다. 경험을 쌓는 부분을 **시뮬레이터**라고 합니다. 그저 시뮬레이터를 1대 더 갖다 놓으면 그만입니다. 혼자서 시행착오를 통해서 배운다면 한참 걸리겠지만 무수히 늘어난 병렬성의 힘 아래에서는 어려운 지식도 빠르게 습득할 수 있게 됩니다. 실제로 OpenAI에서는 Dota2에 강화 학습을 적용할 때에는 시뮬레이션을 위해 256개의 GPU와 12만 8천 개의 CPU코어가 사용되었다고 합니다. 병렬적으로 경험을 쌓는 에이전트의 실제 경험 시간을 모두 합치면 **180년에 해당하는 시간**입니다. 180년 동안 잠도 안 자고, 밥도 안 먹고 24시간 동안 게임만 한 것입니다. 사람이라면 그렇게 할 수 없지만, **기계이기에 이런 방법론이 가능**합니다.

CHAPTER 0 – 강화학습 시작하기

그 유명한 알파고 또한 같은 방법으로 학습되었습니다. 중앙에 뇌에 해당하는 부분이 있고, 이와 연결된 수백, 수천 대의 컴퓨터에서 병렬적으로 바둑을 두는 시뮬레이션이 진행됩니다. 그리고 각 컴퓨터에서 진행된 시뮬레이션 결과는 모두 중앙으로 모여서 **결과적으로 경기에서 이겼던 경우에는 그 경기에서 뒀던 수들을 좀 더 자주 두도록 하고, 졌던 경우에는 해당 경기에서 뒀던 수들을 덜 두도록 조정됩니다.**

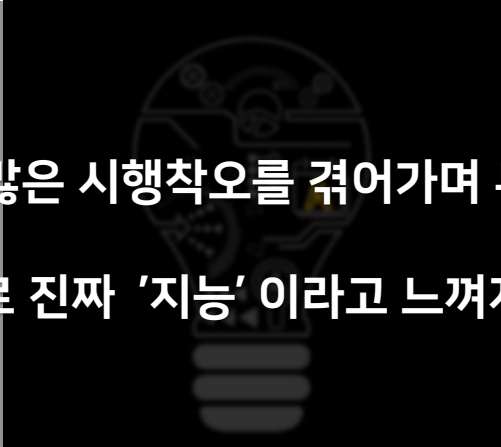
게다가 이 거대한 병렬성은 학습할 때뿐만 아니라 이세돌과 대국을 할 때 실시간으로 주어지는 초읽기 시간에도 마찬가지로 적용되었습니다. 수천 대의 컴퓨터가 다음 수를 몇 수 앞까지 뒤 보면서 실제로 결과가 가장 좋았던 수를 찾아 중앙의 뇌에 해당하는 컴퓨터에 알려주는 겁니다. 이에 대해 보다 자세한 내용은 뒷 부분에서 다뤄보도록 하겠습니다.



하지만 이세돌과 뒀던 알파고의 버전에 비해 그 이후로도 강화 학습을 통해 계속 발전한 최신 버전의 알파고는 더 이상 사람에게 패하지 않는 수준에 이르게 되었습니다. 그러니 병렬성과 조합된 강화 학습의 힘은 매우 강력하단 것을 알 수 있습니다.

▶ 자가 학습(self-learning)의 매력

두 그룹의 학생들이 있다고 하겠습니다. **첫 번째 그룹**은 사교육의 도움을 착실하게 받아오던 학생들입니다. 여러 경로의 사교육을 통해 선행학습이 되어 있으며, 가르쳐 주는 지식을 흡수하는데 특화된 학생들입니다. **두 번째 그룹**은 사교육 보다는 혼자 공부해서 실력을 쌓아오던 학생들입니다. 특별한 지도 없이도 혼자 공부해서 실력을 쌓아 오던 학생들입니다. 이처럼 성향이 다른 그룹의 학생들이 모인 곳에서 과연 전교 1등은 누구의 차지이 될지에 대한 여부는 매우 흥미로운 주제입니다. 놀랍게도 첫 중간고사에서 최고 성적을 기록한 학생은 두 번째 그룹에 속하는 학생이었습니다. 그 학생은 한 번도 사교육을 받아본 적이 없었습니다. 전설처럼 “교과서 위주의 예습과 복습을 철저히” 를 정말로 실천에 옮긴 학생이었습니다. 이 사건으로 **‘가장 빛을 발하는 것은 혼자서 공부할 줄 아는 힘’**에 대한 교훈을 얻게 됩니다.



에이전트는 수 없이 많은 시행착오를 겪어가며 무언가를 깨닫습니다.
이 부분이야말로 진짜 ‘지능’ 이라고 느껴지는 과정입니다.

전교 1등을 차지한 학생의 공부 방식은 조금은 가혹한 강화 학습과 매우 닮아있습니다. **강화 학습**은 그냥 시뮬레이션 환경 속에 던져 놓고 달성해야 할 목적만을 알려주며 알아서 배우라고 합니다. 그러면 **우리의 에이전트는 수 없이 많은 시행착오를 겪어가며 무언가를 깨닫습니다.** 이 부분이야말로 진짜 **‘지능’** 이라고 느껴지는 과정이라 할 수 있습니다.

지도 학습은 무언가 자꾸 정답을 알려주는 반면, 강화 학습은 혼자 놔두고 알아서 성장하길 기다립니다. 그러다 보니 굉장히 유연하고 또 자유롭습니다. 심지어 성능면에서도 뛰어납니다. 지도 학습의 예시로 들었던 에이림의 경우를 생각해 봅시다. 선생님이로부터 자전거를 배운 에이림은 선생님보다 자전거를 더 잘 탈 수 있을까요? **에이림의 학습 과정에서의 목적은 선생님의 지식을 잘 전달받는 것인 만큼 선생님이 모르는 지식을 전달받기는 어렵습니다.** 반면 멧쟁이 사자는 **혼자서 스스로 깨우쳐 가기 때문에 한계란 것이 있을 수 없습니다.** 충분히 다양한 시도를 지속해서 한다면, 어떤 지식이라도 깨우칠 수 있습니다.

알파고도 마찬가지입니다. **알파고 또한 학습 초기에는 프로 바둑기사들의 기보를 통해 지도 학습을 진행**하였습니다. 이 상황에서는 이렇게 되라, 저 상황에서는 저렇게 되라, 사람이 두었던 수를 정답으로 입력받아 그것을 잘 따라 하도록 학습했습니다. 하지만 그렇게만 학습했다면 프로기사를 이길 리는 만무했을 것입니다. **사람을 뛰어넘는 것이 가능했던 이유는 자가 학습에 기반을 둔 강화 학습 덕분**입니다. 승리라는 목표만 알려줬을 뿐 그 과정은 알아서 찾도록 했기 때문에 충분한 **계산 능력(computational power)**과 어우러져 사람이 생각해낼 수 없는 수를 찾아냈던 것이죠.



사람이 알려준 지식을 잘 따라하는 데에 그쳤다면 그것을 위대한 지능이라 부를 수 있었을까요?

그런 면에서 강화학습은 진정한 인공지능에 가까이 있는 어떠한 것이 아닐까 합니다.

마르코프 결정 프로세스

(Markov Decision Process)



CHAPTER 1 – 마르코프 결정 프로세스

이번 장에서는 강화 학습이 풀고자 하는 문제에 대해 좀 더 자세하게 다뤄보겠습니다. 아시다시피 **강화 학습은 순차적 의사결정 문제를 푸는 방법론**이라고 얘기했지만, 사실 아직은 조금 추상적입니다. **순차적 의사결정 문제**는 결국 **MDP(Markov Decision Process)**라는 개념을 통해 더 정확하게 표현할 수 있습니다. 그래서 이번 장의 목적은 MDP가 무엇인지 이해하는 것입니다. 바로 MDP를 설명하면 조금 복잡하게 느껴질 수 있으니 차근차근 단계를 밟아가며 가장 간단한 개념부터 시작하여 조금씩 복잡해질 것입니다. 그래서 먼저 가장 간단한 **마르코프 프로세스(Markov Process)**를 설명하고, **마르코프 리워드 프로세스(Markov Reword Process)**를 설명한 후에 마지막으로 MDP에 대해 설명하겠습니다.

설명의 순서

마르코프 프로세스
(Markov Process)



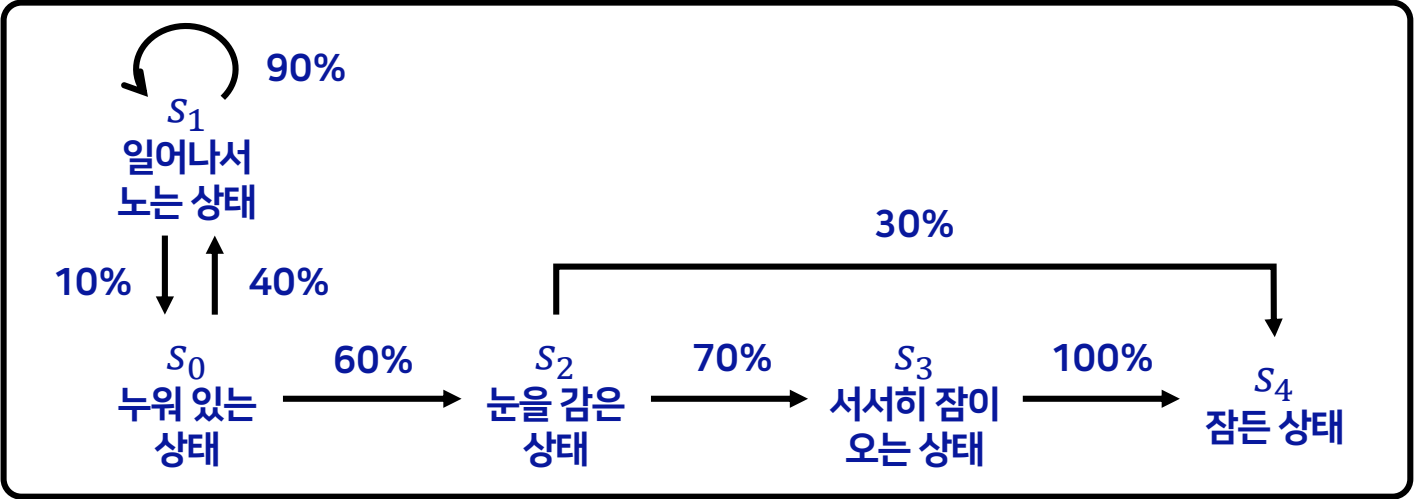
마르코프 리워드 프로세스
(Markov Reword Process)



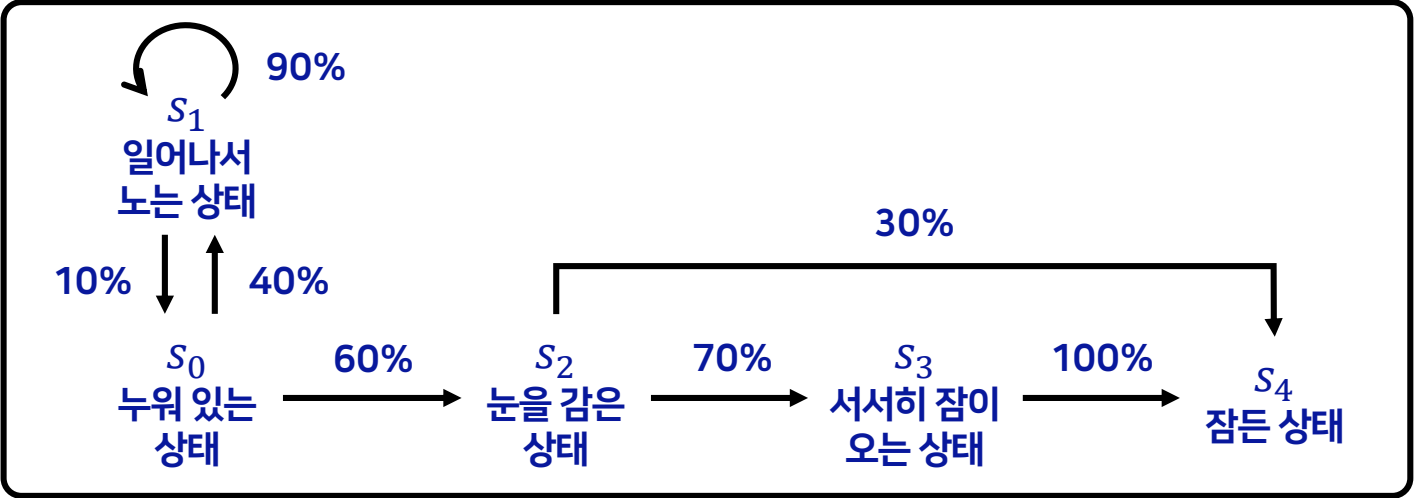
MDP
(Markov Decision Process)

▶ 마르코프 프로세스(Markov Process)

마르코프 프로세스는 이름은 어렵지만 그 자체로는 사실 그렇게 어려운 개념은 아닙니다. 잠에 드는 아이의 상황을 비유하여 마르코프 프로세스를 설명하겠습니다.



다음 그림은 아이가 잠이 들 때 벌어지는 상황을 마르코프 프로세스로 모델링한 것입니다. 아이가 취할 수 있는 **상태(state)**의 종류는 총 5가지 입니다. 누워 있는 상태 s_0 , 일어나서 노는 상태 s_1 , 눈을 감고 있는 상태 s_2 , 서서히 잠이오는 상태 s_3 , 마침내 잠이 든 상태 s_4 입니다. 그리고 아이가 하나의 상태에 진입하게 되면 해당 상태에서 1분씩 머물게 됩니다. 1분이 지나면 다음 상태로 **상태 전이(state transition)**를 합니다. 상태 전이는 현재 상태에서 다음 상태로 넘어간다는 것의 다른 말입니다. 아이가 잠에 드는 과정은 항상 자리에 눕는 것으로 부터 시작하기 때문에 마르코프 프로세스의 첫 상태는 s_0 입니다. 따라서 모든 여정은 s_0 에서 시작합니다.



아이가 누워서 그대로 잠들면 좋겠지만, 아시다시피 그런 일은 흔치 않습니다. 아이는 1분 동안 누워있다가 40%의 확률로 일어나서 노는 상태 s_1 으로 전이하거나, 60%의 확률로 눈을 감은 상태 s_2 로 넘어갑니다. 운이 나쁘게 40%의 확률에 걸려 아이가 일어나면 이제 아이의 상태는 s_1 입니다. 일단 일어나면 지칠 줄 모르는 아이는 쉽사리 s_0 으로 돌아오지 않습니다. 따라서 1분이 지나 **상태 전이**를 해야 할 때, 무려 90%의 확률로 s_1 에 그대로 머물게 됩니다. 그렇게 몇 번이고 같은 상태에 머물다가 운 좋게 s_0 로 돌아오는 데에 성공합니다.

이번에는 다행히 눈을 감은 상태인 s_2 로 상태가 바뀌었고, 그 때부터는 그날 아이가 얼마나 피곤했는지에 따라 다음 상태가 정해집니다. 아이가 무척 피곤했다면 1분간 눈을 감고 있다가 바로 잠든 상태로 넘어가지만 이 확률은 30%입니다. 70%의 확률로는 서서히 잠이 오는 상태인 s_3 로 이동하며 일단 s_3 에 오면 후진은 없습니다. 다음 단계에서 아이는 잠이 들어 s_4 로 이동합니다. s_4 는 **종료 상태(terminal state)**여서 s_4 에 도달하는 순간 마르코프 프로세스는 끝이 납니다.

앞서 보았던 것처럼 마르코프 프로세스는 미리 정의된 어떤 확률 분포를 따라서 상태와 상태 사이를 이동해 다니는 여정입니다. 어떤 상태에 도착하게 되면 그 상태에서 다음 상태가 어디가 될지 각각에 해당하는 확률이 있고, 그 확률에 따라 다음 상태가 정해집니다. 하나의 상태에서 뺏어 나가는 화살표의 합은 항상 100%입니다. 이것이 마르코프 프로세스입니다. 그러면 마르코프 프로세스를 엄밀하게 정의하기 위해서는 어떤 요소들이 필요할까요? 정답은 다음과 같습니다. (※ MP는 Markov Process 의 약자 입니다.)

$$MP \equiv (S, P)$$

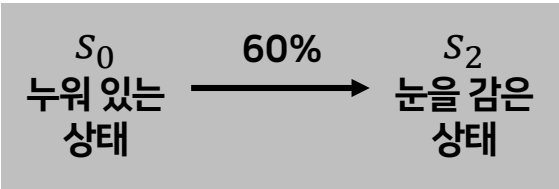
※ *logical representation*
≡ *(is defined as)*

▶ 상태의 집합 S

가능한 상태들을 모두 모아놓은 집합입니다. 아이가 잠드는 마르코프 프로세스의 경우에는 이 집합의 원소가 5개로, $S = \{s_0, s_1, s_2, s_3, s_4\}$ 였습니다.

▶ 전이 확률 행렬 P

$$P_{ss'}$$



전이 확률(*transition probability*) $P_{ss'}$ 은 상태 s 에서 다음 상태 s' 에 도착할 확률을 가리킵니다. 예컨대 위의 마르코프 프로세스에서 s_0 에서 s_2 에 도달할 확률은 60%였습니다. 이를 수식으로 표현하면 $P_{s_0s_2} = 0.6$ 이 됩니다. 첨자가 많아서 어려워 보이지만 사실은 단순합니다.

CHAPTER 1 – 마르코프 결정 프로세스

$P_{ss'}$ 를 **조건부 확률**이라는 개념을 이용해서 조금 다른 방식으로 표현해 볼 수 있습니다.

$$P_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s]$$

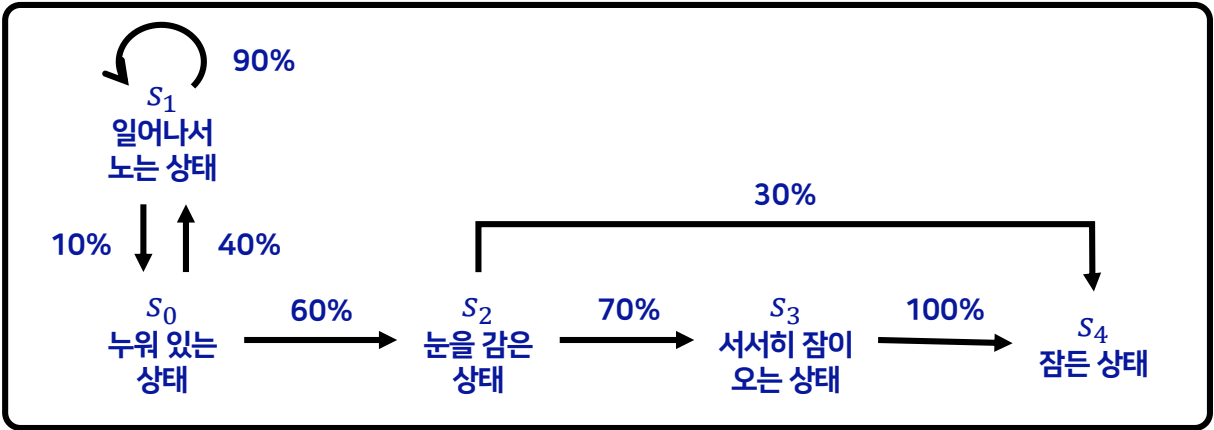
마르코프 프로세스는 정해진 간격(위 예시에서는 1분의 시간 간격)으로 상태가 바뀝니다. 그래서 **시점 t 에서 상태가 s 였다는 것을 조건부 확률의 조건 부분에 집어 넣었습니다. 결국 시점 t 에서 상태가 s 였다면 $t + 1$ 에서의 상태가 s' 이 될 확률**이라는 뜻입니다.

Q. 왜 전이 확률이라 하지 않고 전이 확률 행렬(transition probability matrix)이라고 할까요?

A. $P_{ss'}$ 값을 각 상태 s 와 s' 에 대해 행렬의 형태로 표현할 수 있기 때문입니다.

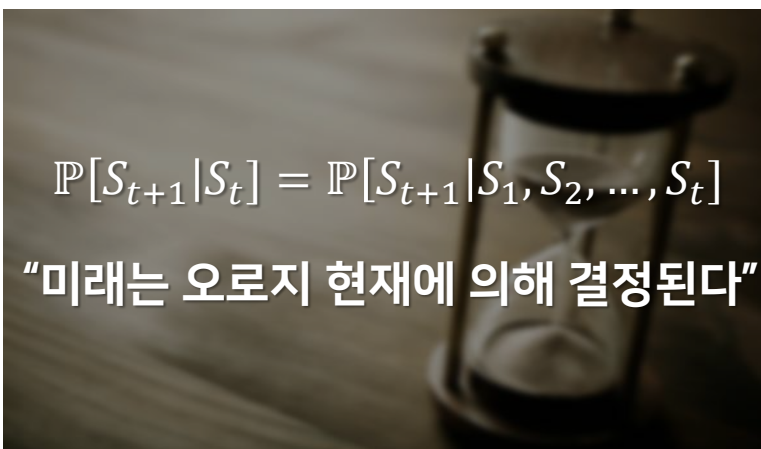
아이가 잠드는 마르코프 프로세스에서 상태의 개수는 총 5개였기 때문에 전이 확률 행렬은 총 25개의 값이 필요합니다. 따라서 $P_{ss'}$ 를 행렬로 표현해 보면 아래와 같은 행렬이 생깁니다.

	누움(s_0)	일어나남(s_1)	눈감음(s_2)	잠이 옴(s_3)	잠듦(s_4)
누움(s_0)		0.4	0.6		
일어나남(s_1)	0.1	0.9			
눈감음(s_2)				0.7	0.3
잠이 옴(s_3)					1.0
잠듦(s_4)					1.0



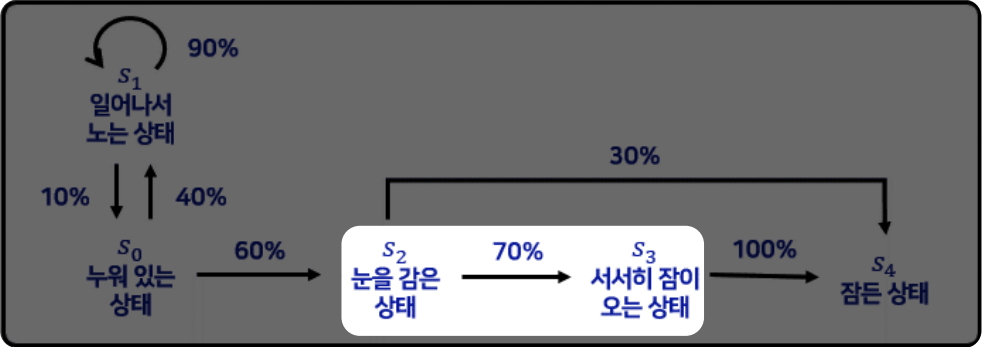
▶ 마르코프 성질

지금까지 마르코프 프로세스가 무엇인지, 마르코프 프로세스를 어떻게 정의하는지에 대하여 살펴보았습니다. 그런데 이 과정의 이름이 왜 “마르코프” 프로세스 일까요? 이 이름에는 중요한 의미가 있습니다. 그 이유는 바로 마르코프 프로세스의 모든 상태가 **마르코프 성질(Markov property)**을 따르기 때문입니다. 마르코프 성질을 따른다는 것의 정의와 뜻은 다음과 같습니다.


$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, \dots, S_t]$$

“미래는 오로지 현재에 의해 결정된다”

보시다시피 정의에 조건부 확률이 사용되고, 조건부 확률에서 바(bar, “|”)의 오른쪽에 주어진 조건을 나타냅니다. 그러므로 마르코프 프로세스에서 **다음 상태가 s_{t+1} 이 될 확률을 계산하려면 현재의 상태 s_t 가 무엇인지만 주어지면 충분**하다는 뜻입니다. s_t 이전에 방문했던 상태들 s_1, s_2, \dots, s_{t-1} 에 대한 정보는 $t + 1$ 시점에 어느 상태에 도달할지 결정하는 데에 아무런 영향을 주지 못합니다. 다시말해 **미래는 오로지 현재에 의해 결정되는 것**입니다.



아이가 잠드는 MP의 예시를 생각해 봅시다. 아이는 현재 s_2 에 있습니다. 여러분은 다음 상태가 s_3 가 될 확률이 70%인지 알고 있을 것입니다. 여기서 아이가 s_2 에 오기 전에 어떤 상태들을 방문했는지 알 수 있을까요? 아마도 모를겁니다. 왜냐면 알려준 적이 없기 때문입니다. 그런데도 여러분은 다음 상태가 s_3 이 될 확률이 70%라는 것을 정확하게 알고 있습니다. 그 이유는 위의 과정이 마르코프 프로세스이기 때문이고, 따라서 모든 상태가 마르코프하기 때문입니다.

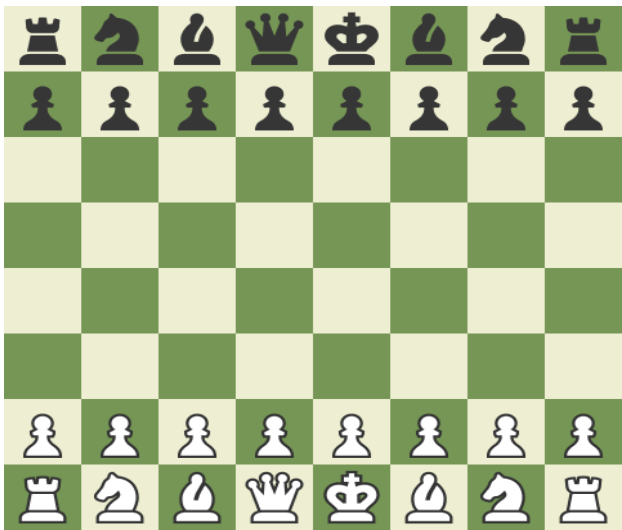
사실 s_2 에 오기까지 정말 다양한 가능성이 있습니다. $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2$ 의 여정을 지나 왔을 수도 있고, $s_0 \rightarrow s_1 \rightarrow s_1 \rightarrow s_1 \rightarrow s_0 \rightarrow s_2$ 의 여정을 지나왔을 수도 있습니다. 하지만 그것은 중요하지 않습니다. 어떤 상태를 어떻게 거쳐서 왔든 다음 상태가 s_3 될 확률이 70%임에는 변함이 없습니다. 아이가 잠드는 과정이 정말 마르코프한 지는 모르겠으나, 여하튼 우리는 이러한 가정을 가지고 아이가 잠이 드는 과정을 마르코프 프로세스로 모델링했던 것입니다. 이처럼 학습을 적용함에 있어서 현재 에이전트에게 주어지고 있는 상태가 마르코프 상태인지 따져보는 것은 이론적으로도, 또 응용의 측면에서도 정말 중요한 과정입니다. 이해를 더하기 위해 마르코프한 상태와 마르코프 하지 않은 상태 각각의 예를 들어보겠습니다.

CHAPTER 1 – 마르코프 결정 프로세스

▶ 마르코프한 상태

먼저 체스 게임을 생각해 보겠습니다. 흑과 백이 치열한 접전을 벌이고 있습니다. 지금은 백의 차례고 최선의 수가 떠올랐습니다. **백이 떠오른 수는 흑과 백이 어떤 상황을 지나서 이 상황에 도달했는지 여부와 관련이 있을까요?**

한 수 이전에 내 말이 어디 있었는지, 상대방의 퀸이 어느 자리에 있었는지, 내 폰이 어디서 잡아 먹혔는지 등의 **과거 이력에 따라 현재 두어야 하는 최선의 수가 바뀔까요?** 짐작하시겠지만 **현재 상황에서 해야 하는 일과 과거 경기 양상이 어떠했는지 여부는 아무런 관련이 없습니다.** 당장 맞닥뜨린 상황을 잘 읽고 미래를 내다보며 최선의 수를 뒤야 하는 것이죠. 따라서 체스 판의 상태를 마르코프 상태라고 할 수 있습니다. 다른 말로 하면 어느 시점 t 에 사진을 찍어서 사진만 보여주고 체스를 두라고 해도 손쉽게 어떤 수를 둘지 정할 수 있습니다.



S_{t-n}

...



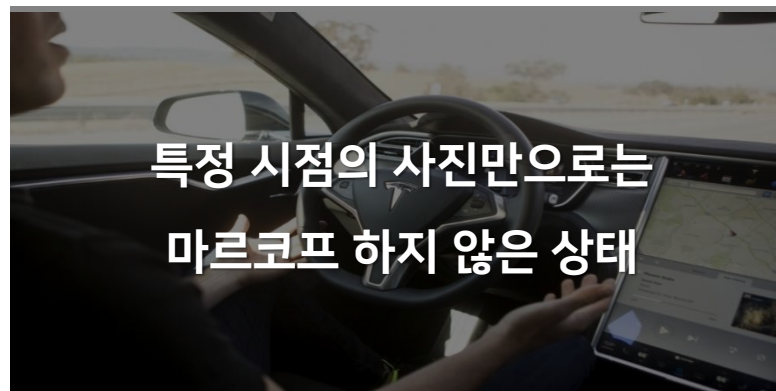
S_t

... S_{t+1}

▶ 마르코프 하지 않은 상태

반면에 운전하고 있는 운전자의 상태를 생각해 봅시다. 특정 시점에 사진을 찍어서 해당 사진만으로도 운전을 해야 한다고 해 봅시다.

과연 여러분들은 이와 같이 사진만 가지고 브레이크를 밟아야 할지, 엑셀을 밟아야 할지 알 수 있을까요? 아마도 어려울 것입니다. 왜냐하면 지금 차가 앞으로 가고 있는지, 뒤로 가고 있는지 알 수 없기 때문입니다. 따라서 사진만 가지고 구성한 운전자의 상태로는 마르코프한 상태가 아닙니다.



Q. 그렇다면 “최근 10초 동안의 사진 10장을 묶어서” 상태로 제공한다면, 이 상태는 마르코프 상태라고 볼 수 있을까요?

A. 그렇습니다. 1초 전의 사진, 2초 전의 사진, ... , 10초 전의 사진이 있다면 이야기는 달라집니다.

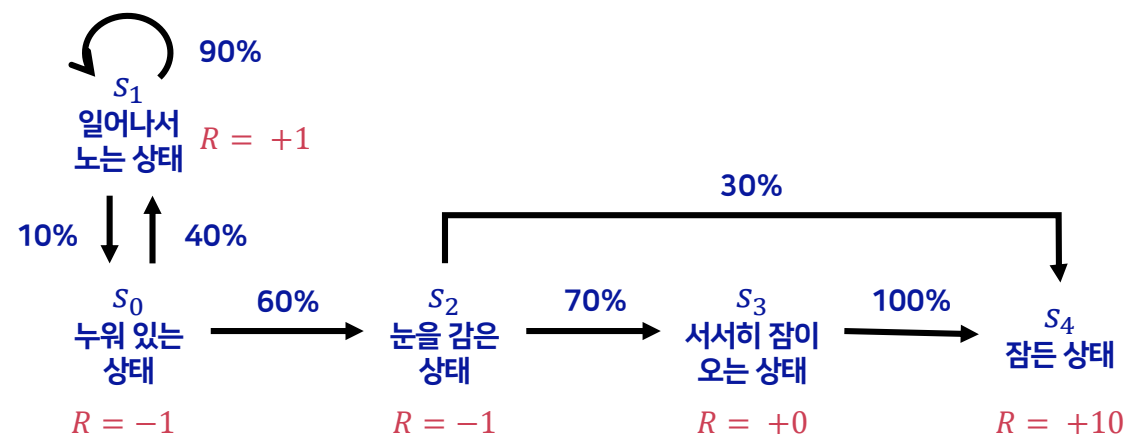
현재 앞으로 가고 있는지, 속도는 몇인지, 가속도는 몇 인지까지 모두 알 수 있기 때문입니다. 완전히 마르코프 하는지는 모르겠지만 적어도 마르코프한 상태에 더 가까워졌다는 것은 의심할 여지가 없을 것입니다. 또 굳이 사진을 묶지 않더라도 특정 시점에 사진과 더불어 속도 벡터, 가속도 벡터 등을 함께 제공한다면 이 또한 마르코프 성질을 더 잘 만족시켜줄 수 있습니다. 실제로 이러한 이유로 딥마인드사에서 발표한 연구에서도 비디오 게임을 플레이하는 인공지능을 학습시킬 때, 시점 t 에서의 이미지와 함께 $t - 1, t - 2, t - 3$ 의 과거 이미지를 엮어서 상태로 제공하였습니다. 이는 모두 상태를 조금이라도 더 마르코프하게 만들기 위함입니다.

요약하자면 [마르코프한 상태](#)도 있고, [마르코프 하지 않은 상태](#)도 있습니다. 어떤 현상을 마르코프 프로세스로 모델링 하려면 상태가 마르코프 해야하며, 단일 상태 정보만으로도 정보가 충분하도록 상태를 잘 구성해야 합니다.

여기까지 마르코프 프로세스의 기본적인 성질들에 대해 살펴보았습니다. 이제 보상이 추가된 [마르코프 리워드 프로세스](#)(Markov Reward Process) 즉 MRP로 넘어가보도록 하겠습니다.

▶ 마르코프 리워드 프로세스(Markov Reward Process)

마르코프 프로세스에 보상의 개념이 추가되면 마르코프 리워드 프로세스가 됩니다. 아래 그림을 보겠습니다.



아까 보았던 아이가 잠이 드는 MP에 빨간색으로 보상 값이 추가된 것을 확인할 수 있습니다. 이제는 어떤 상태에 도착하게 되면 그에 따르는 보상을 받게 되는 것이죠. 예를 들어 자기 위해 가만히 누워 있는 것은 아이 입장에서 조금 답답하기 때문에 -1의 보상을 얻습니다. 반면 일어나서 노는 상태는 당장이 즐겁기 때문에 +1의 보상을 받습니다. 눈을 감게 된 것도, 조금씩 잠이 오는 상태도 각각의 보상을 받으며 마침내 잠들게 되면 목표하던 바를 이루기 때문에 +10의 보상을 받으면서 프로세스는 종료됩니다.

CHAPTER 1 – 마르코프 결정 프로세스

아까 마르코프 프로세스는 상태의 집합 S 와 전이 확률 행렬 P 로 정의 되었는데, MRP를 정의하기 위해서는 R 과 γ (Gamma)라는 2가지 두 가지 요소가 추가로 필요합니다. R 은 보상 함수를 뜻하고 γ 는 감쇠 인자를 가리킵니다. 각각의 요소에 대해 살펴보겠습니다.

$$MRP \equiv (S, P, R, \gamma)$$

▶ 상태의 집합 S

마르코프 프로세스의 S 와 같고, 상태의 집합입니다.

▶ 전이 확률 행렬 P

마르코프 프로세스의 P 와 같고, 상태 s 에서 s' 으로 갈 확률을 행렬의 형태로 표현한 것입니다.

▶ 보상 함수 R

R 은 어떤 상태 s 에 도착했을 때 받게 되는 보상을 의미합니다. 수식으로 표현하면 다음과 같습니다.

$$R = \mathbb{E}[R_t | S_t = s]$$

기댓값이 등장한 이유는 특정 상태에 도달했을 때 받는 보상이 매번 조금씩 다를 수도 있기 때문입니다. 예컨대 어떤 상태에 도달하면 500원짜리 동전을 던져서 앞면이 나오면 500원을 갖고 뒷면이 나오면 갖지 못한다고 할 때, 보상의 값이 매번 바뀌지만 그 기댓값은 250원으로 정해지죠. 아이가 잠이 드는 (※MRP 예시에서는 항상 정해진 보상을 얻는 상황을 가정하였습니다.)

▶ 감쇠 인자 γ

γ 는 0에서 1사이의 숫자입니다. 강화 학습에서 미래 얻을 보상에 비해 **당장 얻는 보상을 얼마나 더 중요하게 여길 것인지를 나타내는 파라미터**입니다. 구체적으로는 미래에 얻을 보상의 값에 γ 가 여러 번 곱해지며 그 값을 작게 만드는 역할을 합니다. 어떤 값을 작게 만들기 때문에 감쇠라는 단어가 쓰였습니다. 이에 대해 자세하게 설명하기 전에 먼저 현재부터 미래에 얻게 될 보상의 합을 가리키는 **리턴(Return)**이라는 개념을 설명하겠습니다. 왜냐하면 γ 는 리턴을 이해해야 그 의미를 진정으로 이해할 수 있습니다.

▶ 감쇠된 보상의 합, 리턴

MRP에서는 MP와 다르게 **상태가 바뀔 때마다 해당하는 보상을 얻습니다**. 상태 s_0 에서 보상 R_0 을 받고 시작하여 종료 상태인 s_T 에 도착할 때 보상 R_T 를 받으며 끝이 납니다. 그러면 s_0 에서 시작하여 s_T 까지 가는 여정을 다음과 같이 표현해 볼 수 있습니다.

$$s_0, R_0, s_1, R_1, s_2, R_2, \dots, s_T, R_T$$

이와 같은 하나의 여정을 강화 학습에서는 에피소드(episode)라고 합니다. 이런 표기법을 이용하여 바로 리턴(*return*) G_t 를 정의할 수 있습니다. **리턴이란 t 시점부터 미래에 받을 감쇠된 보상의 합을 말합니다.**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

보시다시피 현재 타임 스텝이 t 라면 그 이후에 발생하는 모든 보상의 값을 더해줍니다. 또 현재에서 멀어질수록, 즉 **더 미래에 발생할 보상일수록 γ 가 여러 번 곱해집니다.** γ 는 0에서 1사이의 실수이기 때문에 여러 번 곱해질 수록 그 값은 점점 0에 가까워집니다. 따라서 γ 의 크기를 통해 미래에 얻게 될 보상에 비해 현재 얻는 보상에 가중치를 줄 수 있습니다. γ 에 대해서는 뒤에서 좀 더 자세히 다루겠습니다.

리턴은 강화학습에서 정말 중요한 개념입니다. 흔히들 하는 말인 “강화 학습은 보상을 최대화 하도록 학습하는 것이 목적이다.”는 엄밀하게 얘기하자면 완벽한 정답은 아니라 할 수 있습니다. **강화 학습은 보상이 아니라 리턴을 최대화하도록 학습하는 것입니다.** 보상의 합인 리턴이 바로 우리가 최대화하고 싶은 궁극의 목표입니다.

여기서 리턴이 과거의 보상을 고려하지 않고 **미래의 보상을 통해서 정의된다는 것을 유념**해야 합니다. 우리는 과거의 영광에 취해서는 안 됩니다. 시점 t 에 오기까지 그 이전에 100의 보상을 받았건 1000의 보상을 받았건 상관 없습니다. **에이전트의 목적은 지금부터 미래에 받을 보상의 합인 G_t 를 최대화 하는 것입니다.**

▶ 감쇠 인자 γ 는 왜 필요할까?

이제 리턴의 개념을 살펴 보았으니 γ (*gamma*)의 이야기로 돌아오겠습니다. 리턴을 정의할 때 보상을 그냥 더해주는 것이 아니라 감쇠된 보상을 더해줬습니다. γ 는 0에서 1사이의 실수이기 때문에 감마를 여러 번 곱하면 점점 더 0에 가까운 값이 됩니다. 똑같은 +1의 보상이라도 당장 받는 +1의 보상이 100스텝 후에 받는 +1의 보상보다 훨씬 더 큰 값이 된다는 겁니다. 말하자면 미래를 평가 절하해주는 항인 것입니다. 극단적으로 $\gamma = 0$ 이라면 미래의 보상은 모두 0이 되기 때문에 이렇게 학습한 에이전트는 매우 근시안적인 에이전트가 됩니다. 미래는 생각하지 않고 아주 탐욕적(*greedy*)으로 당장의 눈 앞의 이득만 챙기는 것이죠.

반대로 $\gamma = 1$ 이라면 매우 장기적인 시야를 갖고 움직이는 에이전트가 됩니다. 현재의 보상과 미래의 보상이 완전 대등하기 때문입니다.

그렇다면 γ 가 꼭 필요한 이유는 무엇이 있을까요?

그렇다면 γ 가 꼭 필요한 이유는
무엇이 있을까요?

▶ 수학적 편리성

γ 를 사용하는 가장 큰 이유는 γ 를 1보다 작게 해줌으로써 리턴 G_t 가 무한의 값을 가지는 것을 방지할 수 있기 때문입니다. 리턴이 무한한 값을 가질 수 없게 된 덕분에 이와 관련된 여러 이론들을 수학적으로 증명하기가 한결 수월해집니다.



좀 더 쉽게 얘기해보자면 에피소드에서 얻는 각각의 보상의 최댓값이 정해져 있다면, G_t 는 유한하다는 겁니다. 예컨대 MRP를 진행하는 도중 +1, -1, +10 등 다양한 값의 보상을 받을 수 있는데 만일 이 보상이 항상 어떤 상수 값 이하임을 보장할 수 있다면(예를 들어 보상이 아무리 커봐야 +100 보다는 항상 작다 처럼) γ 덕분에 MRP를 무한한 스텝동안 진행하더라도 리턴의 G_t 의 값은 절대 무한한 값이 될 수 없습니다. 이후의 내용에서 차차 다루겠지만 우리는 어떤 상태로 부터 리턴을 예측하고자 합니다. 이때 리턴이 무한이라면 어느 쪽이 더 좋을지 비교하기도 어렵고, 그 값을 정확하게 예측하기도 어려워 집니다. 감마가 1보다 작은 덕분에 이 모든것이 가능해 집니다.

▶ 사람의 선호 반영

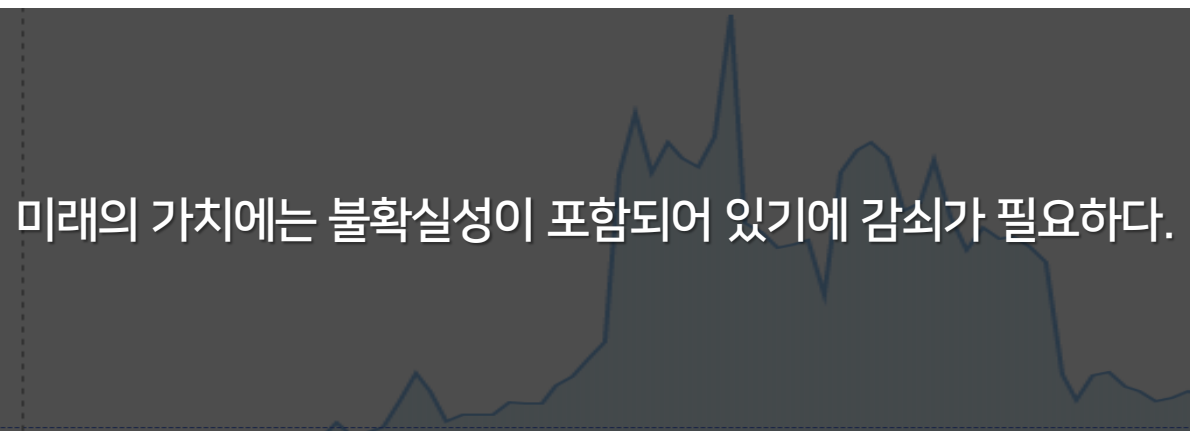


여러분들에게 지금 당장 5만원권 20장, 총 100만원을 드린다고 생각하면 여러분의 기분은 무척 좋을 것입니다. 하지만 5년 후에 100만원을 드린다고 약속하면 지금 당장 받는 것 보다는 조금 덜 좋을 것입니다.

이유야 다양하겠지만 누군가는 100만원을 받아서 지금 당장 주식에 넣어두면 5년 후에는 더 큰 돈이 될 테니 지금 당장 100만원을 받는 것을 선택할 것입니다. 그만큼 사람은 기본적으로 당장 벌어지는 눈 앞의 보상을 더 선호한다는 것을 알 수 있습니다.

에이전트도 마찬가지입니다. 이런 이유에서 에이전트를 학습하는데 있어서 감마라는 개념이 도입하게 됩니다.

▶ 미래에 대한 불확실성 반영



미래는 불확실성 투성입니다. 위의 예시에서 아무리 안전한 주식이라 할지라도 언제나 오를거라는 확신은 할 수 없습니다. 이처럼 현재와 미래 사이에는 다양한 확률적 요소들이 있고 이로 인해 당장 느끼는 가치에 비해 미래에 느끼는 가치가 달라질 수 있습니다. 그렇기 때문에 미래의 가치에는 불확실성을 반영하고자 감쇠를 해줍니다.

▶ MRP에서 각 상태의 벨류 평가하기

여기서 간단하게 질문을 하나 해보겠습니다.

Q. 만일 어떤 상태의 가치를 평가하고 싶다면 어떤 값을 사용하면 좋을까요?

예를 들어 아이가 잠드는 MRP에서 눈을 감고 있는 상태 s_2 의 벨류(value) 혹은 가치를 숫자 하나로 딱 평가하고 싶다면 말이죠.

A. 가장 먼저 떠오르는 생각은 일단 보상이 높을수록 좋은 상태일 것이라는 점입니다.

그런데 s_2 에 도달하기까지 그 이전에 받은 보상이 중요할까요? 아니면 이후에 받을 보상이 중요할까요? 즉, 어떤 상태를 평가하는데 있어서 과거가 중요할까요 미래가 중요할까요? 이는 조금만 생각해보면 비교적 쉽게 답할 수 있습니다. 예를 들어 사람들이 선망하는 기업인 구글에 입사하는 것의 가치를 평가해 봅시다. 그것은 구글에서 앞으로 받을 연봉이 매우 높기 때문일까요, 아니면 구글에 입사하기까지 많은 돈을 받아서(?) 였을까요.

후자는 문장이 성립되기 어려울 정도입니다. 어떤 상태를 평가할 때에는 당연히 그 시점으로 부터 미래에 일어날 보상을 기준으로 평가해야 합니다. 그러니 아이가 잠드는 MRP에서 아이가 눈을 감고 있는 상태를 평가하고자 한다면 마찬가지로 해당 상태를 지나 미래에 받을 보상들을 더해야 합니다.

우리는 마침 그에 해당하는 값을 알고 있습니다. 바로 리턴입니다. 리턴은 그 시점부터 이후에 받을 보상들을 (감쇠하여) 더한 값입니다.

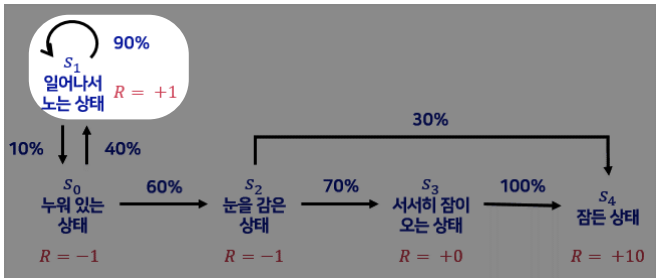
그러면 s_2 의 가치를 평가하고 싶다면 s_2 부터 시작하여 리턴을 측정하면 됩니다.

하지만 한 가지 문제가 있습니다. 그 리턴이라는 값이 매번 바뀐다는 점입니다.

왜냐하면 MRP 자체가 확률적인 요소에 의해 다음 상태가 정해지기 때문에 같은 s_2 에서 출발해도 아이가 잠들 때까지 각기 다른 상태를 방문하며 그때마다 얻는 리턴의 값은 달라지기 때문입니다. 그러면 s_2 의 가치는 어떻게 평가해야 할까요? 정답은 리턴의 기댓값(Expectation)을 사용하는 겁니다. 이 과정을 계속 진행하기 전에 “에피소드를 샘플링한다.”라는 개념을 설명하고, 그에 따라 정말 매번 다른 리턴값을 얻게 되는지를 확인하고 넘어가겠습니다.

▶ 에피소드 샘플링

시작 상태 s_0 에서 출발하여 종료 상태 s_T 까지 가는 하나의 여정을 에피소드라고 했습니다. 그런데 하나의 에피소드 안에서 방문하는 상태들은 매번 다릅니다.

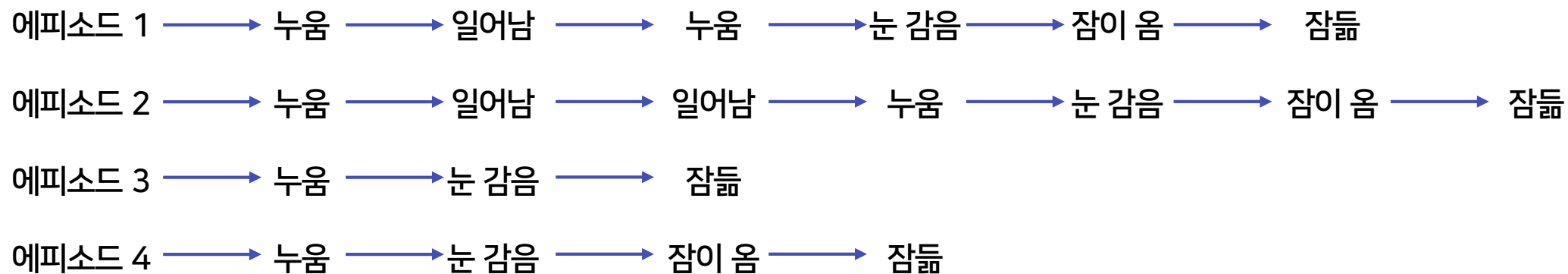


예를 들어 상태 s_1 은 방문할 수도 있고, 방문하지 않을 수도 있습니다. 그리고 그에 따라 리턴도 달라집니다. 이를 좀 더 강화학습에서 사용되는 용어를 빌려 얘기해보자면 매번 에피소드가 어떻게 샘플링(sampling)되느냐에 따라 리턴이 달라집니다. 그렇다면 샘플링은 무엇일까요?

우리말로 표본 추출이라고 하지만, 단어가 너무 어렵고 추상적이어서 그냥 샘플링이라는 표현을 사용합니다. 샘플링이란 sample에 ing가 더해져서 샘플을 뽑아본다는 뜻을 가집니다. 어떤 확률 분포가 있을 때 해당 분포에서 샘플을 뽑아보는 것이 샘플링입니다.

강화학습에서 샘플링은 매우 중요합니다. 동전 던지기의 예시는 확률분포가 간단하지만, 많은 경우에서 실제 확률 분포를 잘 모르는 경우가 대부분입니다. 따라서 우리는 샘플링을 통해서 어떤 값을 유추하는 방법론을 사용하게 됩니다. 이러한 류의 방법론을 일컬어 Monte-Carlo 접근법이라고 부릅니다. 이에 대해서 앞으로 더 자세하게 배워보겠습니다.

정리하자면 위와 같은 샘플링 기법을 통해 주어진 MRP에서 여러 에피소드를 샘플링해 볼 수 있습니다. 예컨대 아이 재우기 MRP에서는 상태마다 다음 상태가 어떻게 될지에 대한 확률 분포가 주어져 있습니다. 이때 각 상태에서 마치 동전 던지기와 같은 과정을 거쳐서 다음 상태가 정해집니다. 각 상태마다 다음 상태를 샘플링하며 진행하면 언젠가 종료 상태에 도달할 것이고 하나의 에피소드가 끝이 납니다. 이제 다음 에피소드를 시작하여 이와 같은 작업을 반복해볼 수 있고, 우리는 아래와 같은 에피소드의 샘플들을 얻게 됩니다.

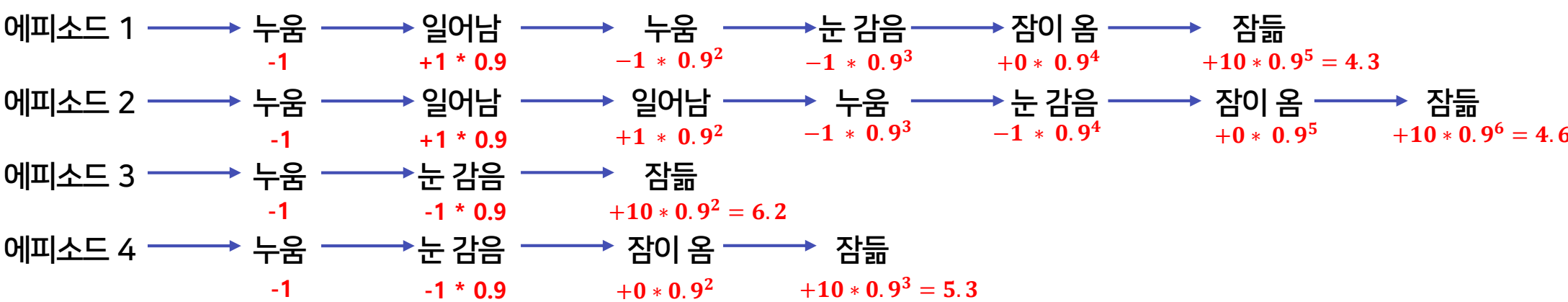


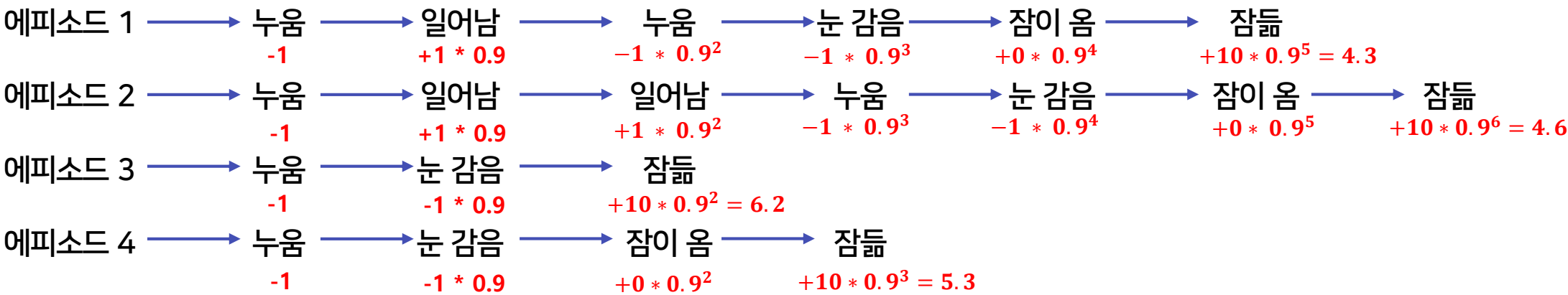
▶ 상태 가치 함수(State Value Function)

우리의 문제 의식은 상태를 어떻게 하면 주어진 상태 s 의 가치를 평가할 수 있을까? 에서 출발하였습니다. 그래서 상태 가치 함수라는 것을 가정해 보겠습니다. 상태 가치 함수는 상태를 인풋으로 넣으면 그 상태의 가치를 아웃풋으로 출력하는 함수입니다. 앞서 살펴본 것 처럼 에피소드마다 리턴이 다르기 때문에 어떤 상태 s 의 벨류 $v(s)$ 는 기댓값을 이용하여 정의합니다.

$$v(s) = \mathbb{E}[G_t | S_t = s]$$

조건부로 붙는 $S_t = s$ 의 의미는 시점 t 에서 상태 s 부터 시작하여 에피소드가 끝날 때 까지의 리턴을 계산하라는 뜻이 됩니다. 이해를 돕기 위해 아이가 잠드는 MRP에서 상태 s_0 의 벨류에 대해 생각해 보겠습니다. 아래와 같이 s_0 에서 시작하는 에피소드 4개를 샘플링하여 그때마다 리턴을 계산해봅니다.





위 그림과 같이 s_0 에서 출발하여 발생할 수 있는 에피소드는 무한히 많고, 그때마다 리턴도 항상 다릅니다. 기댓값을 구하려면 에피소드별로 해당 에피소드가 발생할 확률과 그때의 리턴 값을 곱해서 더해줘야 합니다. 가능한 에피소드가 무한히 많기 때문에 이런 접근법은 현실적으로 불가능합니다. 그래서 간단한 방법은 샘플로 얻은 리턴의 평균을 통해 벨류를 근사하게나마 계산해 볼 수 있습니다. 예컨대 위의 그림에서는 4.3, 4.6, 6.2, 5.4의 평균인 5.1이 $v(s_0)$ 가 됩니다.

자, 지금까지 배운 내용들을 잠깐 정리하고 넘어가겠습니다. 지금까지 MP는 (S,P) 로 정의되었고, MRP는 보상이 추가되어 (S,P,R,γ) 로 정의할 수 있습니다. 현재 시점부터 받을 보상의 합이 리턴임을 배웠고, 같은 상태에서 출발하여도 에피소드마다 리턴이 달라지므로 주어진 상태의 가치를 리턴의 기댓값을 통해 정의할 수 있다는 것도 배웠습니다.

이제 마지막 목표인 MDP에 대한 내용으로 넘어가겠습니다. MP, MRP는 MDP를 설명하기 위한 준비운동이었다고 생각해도 좋습니다.

▶ 마르코프 결정 프로세스(Markov Decision Process)

생각해보면 앞서 배웠던 MP와 MRP에서는 상태 변화가 자동으로 이루어졌습니다. 다음 상태의 분포는 미리 정해져 있었죠. 여기에는 행동하는 주체랄 것이 없었습니다. 아이가 깨고 잠다는 예시 또한 미리 정해진 확률에 따라 정해지는 과정이니 마치 흐르는 강물 위에 몸을 맡긴 것과 같았습니다. **따라서 MP나 MRP만 가지고는 순차적 의사결정 문제를 모델링할 수 없습니다.** 순차적 의사결정에는 **의사결정(decision)**이 핵심이기 때문입니다. 그래서 의사 결정에 관한 부분이 모델에 포함되어 있어야 합니다. 바로 그런 이유에서 **마르코프 결정 프로세스(Markov Decision Process)**가 등장합니다. 이제 자신의 의사를 가지고 행동하는 주체인 에이전트가 등장합니다.

▶ MDP의 정의

MDP는 MRP에 에이전트가 더해진 것입니다. 에이전트는 각 상황마다 액션(행동)을 취합니다. 해당 액션에 의해 상태가 변하고 그에 따른 보상을 받습니다. 이 때문에 MDP를 정의하기 위해서는 하나의 요소가 추가되니, 그것은 바로 **액션의 집합 A** 입니다. **A 가 추가되면서 기존의 전이 확률 행렬 P 나 보상 함수 R 의 정의도 MRP에서의 정의와 약간씩 달라지게 됩니다.** 정리하면 MDP의 구성요소는 **(S, A, P, R, γ)** 입니다.

$$MDP \equiv (S, A, P, R, \gamma)$$

▶ 상태의 집합 S

$$MDP \equiv (S, A, P, R, \gamma)$$

마르코프 프로세스(MP), 마르코프 보상 프로세스(MRP)에서의 S 와 같습니다. 가능한 상태의 집합입니다.

▶ 액션의 집합 A

MDP에서 새롭게 추가된 항목이며 에이전트가 취할 수 있는 액션들을 모아놓은 것입니다. 예를 들어 화성의 흙을 채집하기 위해 보내진 탐사 로봇이 할 수 있는 행동이 “앞으로 움직이기”, “뒤로 움직이기”, “흙을 채집하기” 이렇게 3가지 라면 액션의 집합 A 는 아래와 같습니다.

$$A = \{\text{앞으로 움직이기, 뒤로 움직이기, 흙을 채집하기}\}$$

이제 에이전트는 스텝마다 액션의 집합 중에서 하나를 선택하여 액션을 취하며, 그에 따라 다음에 도착하게 될 상태가 달라집니다.

(※ 이는 이후에 아이 재우기 MDP를 보면 더 잘 이해가 될 것입니다.)

▶ 전이 확률 행렬 P

MP, MRP에서는 “현재 상태가 s 일 때 다음 상태가 s' 이 될 확률”을 $P_{ss'}$ 이라고 표기했습니다. 하지만 MDP에서는 **에이전트가 선택한 액션에 따라서 다음 상태가 달라집니다.** 따라서 “현재 상태가 s 이며 에이전트가 액션 a 를 선택했을 때 다음 상태가 s' 이 될 확률”을 정의해야 합니다.

여기서 주의해야할 점은 상태 s 에서 액션 a 를 선택했을 때 도달하게 되는 상태가 결정론적이지 아니라는 점입니다. 같은 상태 s 에서 같은 액션 a 를 선택해도 매번 다른 상태에 도착할 수 있습니다. 말하자면 액션 실행 후 도달하는 상태 s' 에 대한 확률 분포가 있고 그게 바로 전이 확률 행렬 P 입니다.

예를 들어 바람이 세게 부는 징검 다리를 생각해 보겠습니다. 앞으로 한 걸음 내딛는 액션을 했을 때 바람이 안 분다면 바로 앞에 칸에 도착하겠지만, 바람이 세게 분다면 넘어져서 물에 빠질 수도 있습니다. 즉, **같은 상태에서 같은 액션을 해도 확률적으로 다음 상태가 달라질 수 있습니다**. 바람은 환경의 일부, 즉 MDP의 일부이기 때문에 에이전트가 바꿔줄 수 있는 부분이 아닙니다. 따라서 이러한 확률적 요소를 표현하기 위해 전이 확률이 $P_{ss'}^a$ 의 형태로 표기됩니다.

조건부 확률의 개념을 이용한 $P_{ss'}^a$ 의 엄밀한 정의는 다음과 같습니다.

$$P_{ss'}^a = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a]$$

위 식을 보면 2가지 조건이 붙습니다. **"현재 상태 s 에서, 액션 a 를 했을 때"**가 조건으로 붙는 것입니다.

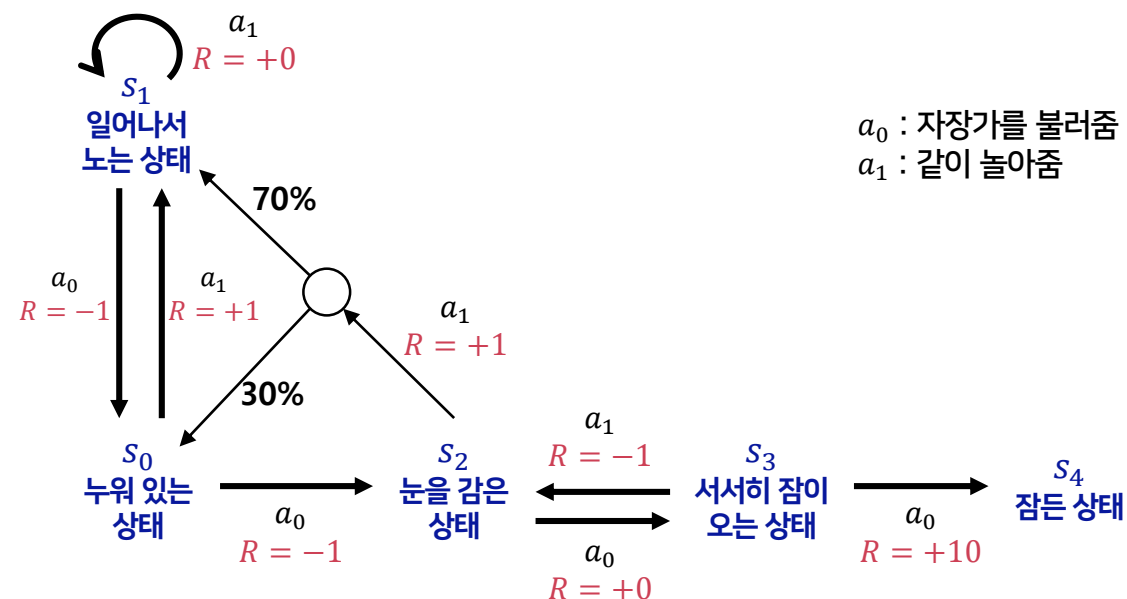
▶ 보상 함수 R

MRP에서는 상태에 의해 보상이 정해졌습니다. 하지만 **MDP에서는 액션이 추가되었기 때문에 현재 상태 s 에서 어떤 액션을 선택하느냐에 따라 받는 보상이 달라집니다**. 이를 반영하기 위해 R_s 에도 액션을 가리키는 a 의 첨자가 하나 붙어서 R_s^a 의 형태가 됩니다. **상태 s 에서 액션 a 를 선택하면 받는 보상**을 가리키며, 이는 확률적으로 매번 바뀔 수 있기 때문에 **마찬가지로 기댓값을 이용하여 표기합니다**. 따라서 MDP에서의 보상함수 R_s^a 의 정의는 다음과 같습니다.

$$R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

▶ 감쇠 인자 γ

감쇠 인자는 MRP에서의 γ 와 정확히 같습니다.

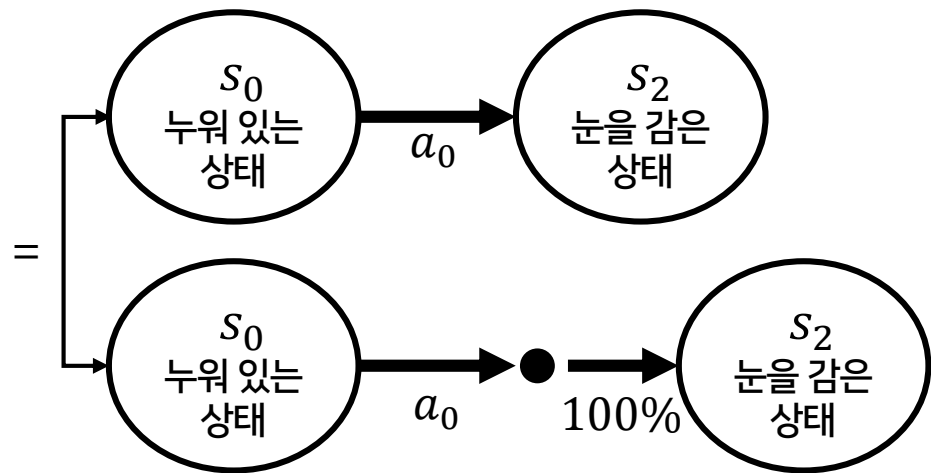


지금까지 MDP를 정의하기 위해 필요한 요소에 대해 알아보았습니다. 아이 재우기 MDP의 예시를 통해 좀 더 구체적으로 설명해 보겠습니다. 아이가 잠드는 상황에서 **어머니라는 에이전트**가 개입되기 시작했습니다. 어머니가 선택할 수 있는 액션은 **자장가를 불러주는 액션 a_0** 과 **함께 놀아주는 액션 a_1** 2가지가 있습니다. 아이가 비교적 활발한 상태인 s_0, s_1 에서는 자장가를 불러주는 액션을 선택하면 당장은 음의 보상을 받습니다. 그 이유는 아이는 기본적으로 일어나서 놀고 싶어하는 상태이기 때문입니다. 반대로 함께 놀아주는 액션을 하면 아이와 함께 즐거운 시간을 보내게 되어 양의 보상을 받습니다. **자장가를 불러 주어서 아이가 잠이 오는 상태 s_3 에 도달하였다면 여기서 같이 놀아주는 행위는 오히려 역효과를 낳아 음의 보상을 받습니다.** 반대로 아이에게 자장가를 불러주면 마침내 아이는 잠에 들게 되고, 어머니의 목표는 성공하게 되어 큰 보상을 받습니다. 눈여겨 볼 부분은 아이가 눈을 감은 상태인 s_2 에서 아이에게 놀아주는 액션을 선택하면 아이의 다음 상태는 그날 아이의 상태에 따라 s_0 가 될 수도 있고, s_1 이 될 수도 있습니다. 이때의 전이 확률을 수식으로 표현하면 아래와 같습니다.

$$P_{s_2, s_0}^{a_1} = 0.3, \quad P_{s_2, s_1}^{a_1} = 0.7$$

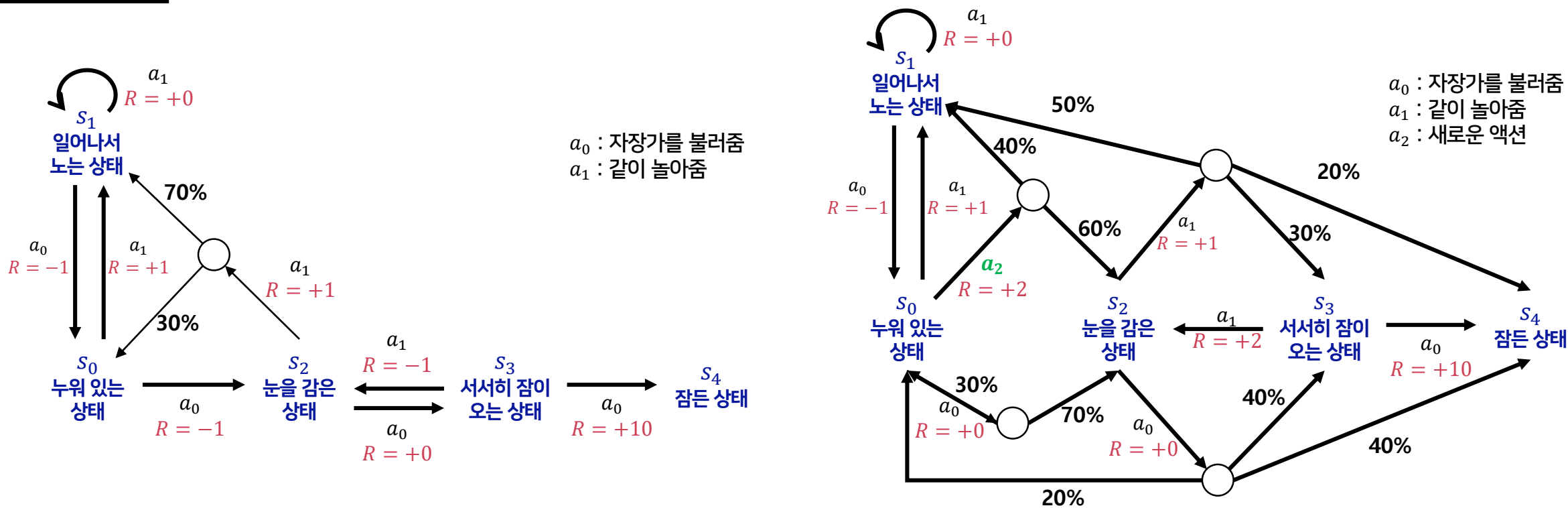
CHAPTER 1 – 마르코프 결정 프로세스

아이의 상태는 환경의 일부이며, 에이전트인 어머니가 조절할 수 있는 것은 아닙니다. 따라서 확률 값을 바꿀 수는 없습니다.



혹시 ‘왜 s_2 에서 자장가를 불렀을 때만 전이 확률이 있고 나머지 상황에서는 전이 확률이 없지?’ 하고 궁금한 분이 있을 수도 있습니다. 이는 위의 그림을 보면 이해가 됩니다. 예컨데 s_0 에서 a_0 를 선택하면 그 뒤에 전이 확률이 100%인 화살표가 하나 더 그려져 있다고 생각해도 됩니다. 편의상 100%인 전이 확률 화살표를 그림에서 생략하였습니다.

그럼 이제 MDP의 기본에 대해 설명을 마쳤으니 강화 학습의 핵심 질문을 던져 보겠습니다. 아이 재우기 MDP에서는 상황마다 어머니가 하는 선택에 따라 보상이 달라지는데, **보상의 합을 최대화하기 위해서 어머니는 어떤 행동을 선택해야 할까요?**



자세히 관찰해보면 어렵지 않게 계속해서 a_0 만을 선택하면 최적이라는 것을 알 수 있습니다. 그렇게 되면 얼마 지나지 않아 아이는 잠드는 상태에 도달하여 큰 보상을 받고 에피소드가 끝나기 때문입니다. 그것이 어머니의 최적의 전략입니다. 이처럼 MDP가 간단한 경우 상태와 관계없이 같은 액션만 선택하면 되기 때문에 최적의 전략을 찾기 쉬워졌습니다. 하지만 MDP가 복잡해지면 최적 행동을 찾는 것이 그렇게 쉽지만은 않습니다. 예컨대 위와 같은 그림처럼 말이죠. 전이 확률이 조금 달라졌고, 상태 s_0 에서 선택할 수 있는 액션이 1개 늘어났을 뿐인데 이전 MDP에 비해 그림이 많이 복잡해졌습니다. 여기서도 누적 보상을 최대화 하기 위해 상황별로 어떤 선택을 해야 할지 쉽게 감이 오지 않습니다. 나름 복잡한 MDP가 만들어 졌으나 실제 생활에서 맞닥뜨리게 되는 순차적 의사결정 문제를 모델링한 것에 비하면 무척이나 간단한 모형입니다.

위 MDP의 경우 상태의 개수가 5개, 액션의 개수가 많아야 3개 정도입니다. 반면 실제 세계에서 마주하는 MDP는 상태의 개수가 수백억 개가 넘을 정도로 무수히 많고, 액션의 개수도 훨씬 많습니다. 이처럼 복잡한 MDP에서 결국 우리가 찾고자 하는 것은 각 상태 s 에 따라 어떤 액션 a 를 선택해야 보상의 합을 최대로 할 수 있는가 입니다. 이것을 위해서는 전략이라고 표현할 수도 있으나 강화 학습에서는 이를 정책(Policy)이라고 합니다.

▶ 정책 함수와 2가지 가치 함수

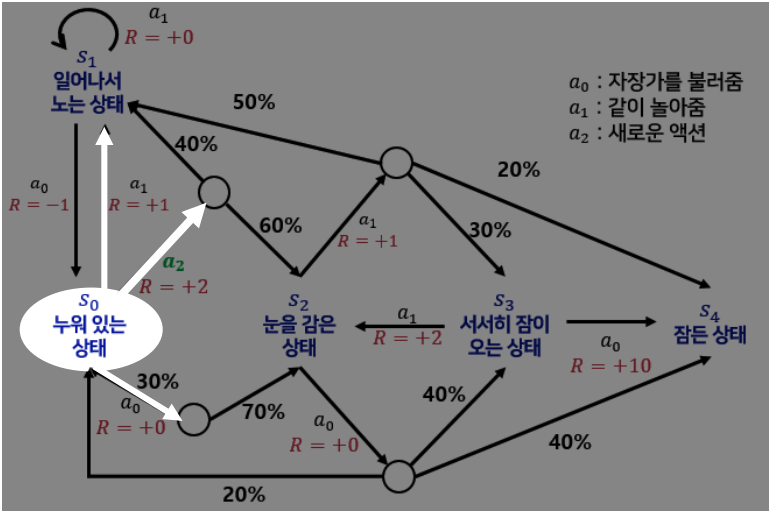
정책 혹은 정책 함수(policy function)는 각 상태에서 어떤 액션을 선택할지 정해주는 함수입니다. 예컨대 아이를 재우려는 어머니의 입장에서 아이의 상태에 따라 a_0 를 선택할지, a_1 를 선택할 지 결정해야 합니다. 그것을 어머니의 정책이라 부를 수 있습니다. 어머니의 목적은 보상의 합을 최대화하는 정책을 찾는 것입니다. 그리고 이 정책을 함수로 표현하면 정책 함수가 됩니다. 정책 함수는 보통 그리스 문자 π 를 사용해서 표기하는데, 이는 원주율 3.14..와는 아무런 관계가 없습니다. 정책 함수를 확률을 이용하여 정의하면 아래와 같습니다.

$$\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$$

(상태 s 에서 액션 a 를 선택할 확률)

CHAPTER 1 – 마르코프 결정 프로세스

예를 들어 아래 그림의 MDP 속 상태 s_0 에서 선택할 수 있는 액션은 a_0, a_1, a_2 이렇게 세 가지 입니다. 그리고 이 각각에 대해 얼마큼의 확률을 부여할지를 정책함수가 결정합니다. 예를 들어 다음과 같이 말이죠.



$$\pi(a_0|s_0) = 0.2$$
$$\pi(a_1|s_0) = 0.5$$
$$\pi(a_2|s_0) = 0.3$$

각 상태에서 할 수 있는 모든 액션의 확률 값을 더하면 1이 되어야 합니다. 꼭 위와 같이 확률이 3개의 액션에 퍼져 있을 필요는 없고, 하나의 액션에 100%의 확률이 모두 몰려있는 것도 가능한 정책입니다. 여하간 에이전트가 각 상태에 처했을 때 각각의 액션을 어떻게 선택할 것인지에 대한, 즉 액션 선택에 대한 운용 방침을 담고 있고 그래서 이를 정책이라고 부릅니다.

CHAPTER 1 – 마르코프 결정 프로세스

한 가지 헛갈리면 안 되는 것이 있습니다. 정책 함수는 에이전트 안에 존재하는 점입니다. 즉 **상태별** 액션을 얼마큼의 확률로 고를지에 대한 내용을 주지 않습니다.

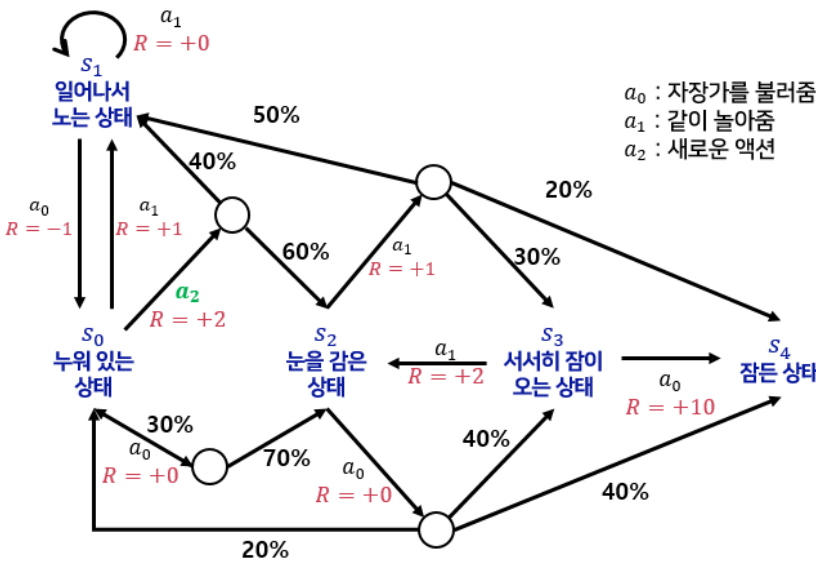
환경은 변하지 않지만 에이전트는 자신의 정책을 언제든지 수정할 수 있습니다. 더 큰 보상을 얻기 위해 계속해서 정책을 교정해 나가는 것이 곧 강화학습입니다.

▶ 상태 가치 함수

MRP에서 배웠던 **가치 함수**를 떠올려 보겠습니다. 가치 함수의 정의는 **어떤 상태를 평가하고 싶다는 의도**에서 출발하였습니다. 주어진 상태에서 미래에 얻을 리턴의 값을 상태의 벨류로 정의했었습니다. MDP에서의 **가치 함수**도 비슷하지만 약간의 차이가 있습니다. 왜냐하면, **에이전트의 액션이 도입**되었기 때문입니다. 이제는 에이전트가 어떤 행동을 취하는가에 따라, 더 정확히 표현하면 **에이전트의 정책 함수에 따라서 얻는 리턴이 달라**집니다. 예를 들어 아이 재우기 MDP에서 항상 a_0 를 선택하는 에이전트와 항상 a_1 을 선택하는 에이전트의 누적 보상은 분명 다를 것입니다. 따라서 **가치 함수는 정책 함수에 의존적**입니다. **가치 함수를 정의**하기 위해서는 먼저 **정책 함수 π 가 정의**되어야 합니다. π 가 주어졌다고 가정했을 때 가치 함수의 정의는 다음과 같습니다.

$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid S_t = s] \\ &= \mathbb{E}_{\pi}[G_t \mid S_t = s] \end{aligned}$$

s 부터 끝까지 π 를 따라서 움직일 때 얻는 **리턴의 기댓값**



$$\begin{aligned} v_{\pi}(s) &= \mathbb{E}_{\pi}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | S_t = s] \\ &= \mathbb{E}_{\pi}[G_t | S_t = s] \end{aligned}$$

식에 정책을 나타내는 π 가 추가되었을 뿐 나머지는 원래 가치 함수의 정의와 똑같습니다. 이 식은 정책 함수를 π 로 고정해놓고 생각하겠다는 것입니다. 지금부터 이후에 나올 모든 가치 함수는 MDP에서의 가치 함수만 다를 것이기 때문에 MRP에서의 가치 함수의 정의는 잠시 잊어도 좋습니다. 이제부터 가치 함수는 π 에 의존적이라는 것을 다시한번 명심하는 것이 좋습니다.

▶ 액션 가치 함수

앞에서 배운 가치 함수는 말하자면 상태 가치 함수입니다. 어떤 상태가 주어졌을 때 그 상태를 평가해주는 함수입니다. 그러면 자연스럽게 다음과 같은 궁금증이 떠오릅니다.

“각 상태에서의 액션도 평가할 수는 없을까?”

상태를 평가할 수 있었다면 해당 상태의 액션도 평가 할 수 있지 않을까요? 액션을 평가할 수 있다면 각 상태에서 선택할 수 있는 액션을 모두 평가 해 본 다음에, 그중에 가장 가치있는 액션을 선택하면 될 테니까요. 그런 이유에서 액션 가치 함수(state-action value function)가 등장합니다. 정식 명칭은 상태-액션 가치 함수 이지만 축약하여 액션 가치 함수라고 부르겠습니다.

상태 가치 함수를 $v(s)$ 로 표현하였다면 액션 가치 함수는 $q(s, a)$ 로 표현합니다. 함수에 상태와 액션이 동시에 인풋으로 들어가야 합니다. 그래서 정식 명칭이 상태-액션 가치 함수입니다. 그렇다면 왜 액션만 따로 떼어내어 평가할 수는 없을까요?

그 이유는 상태에 따라 액션의 결과가 달라지기 때문입니다. 예를 들어 앞의 그림에서 s_0 에서 a_0 를 선택하는 것과 s_2 에서 a_0 를 선택하는 것은 전혀 다른 상황입니다. 이후에 도착하는 상태도 다르고, 이후에 얻는 보상도 다릅니다. 그래서 액션 a 만을 따로 평가할 수는 없고 상태 s 와 결합하여 평가해야 합니다. 요컨대 액션 가치 함수는 상태와 액션의 페어, 즉 (s, a) 를 평가하는 함수입니다.

여기까지가 차이이고, 나머지 상태 가치 함수 $v(s)$ 와 거의 똑같습니다. $v(s)$ 와 마찬가지로 미래에 얻을 리턴 G_t 를 통해 밸류를 측정합니다. 또한 정책 π 에 따라 이후의 상태가 달라지므로 π 를 고정시켜 놓고 평가합니다. 이를 수학적으로 표현하면 다음과 같습니다.

$$q_{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | S_t = s, A_t = a]$$

s 에서 a 를 선택하고, 그 이후에는 π 를 따라 움직일 때 얻는 리턴의 기댓값

보다시피 함수에 인풋으로 상태 s 와 선택한 액션 a 가 함께 들어가며, 아웃풋으로 s 에서 a 를 선택하는 것의 밸류가 나옵니다. 일단 a 를 선택하고 나면 이후의 상태를 진행하기 위해 계속해서 누군가가 액션을 선택해야 하는데, 그 역할은 정책 함수 π 에게 맡기는 겁니다.

따라서 $v_{\pi}(s)$ 와 $q_{\pi}(s, a)$ 는 “ s 에서 어떤 액션을 선택하는가?” 하는 부분에만 차이가 있습니다. $v_{\pi}(s)$ 를 계산할 때는 s 에서 π 가 액션을 선택하는 반면, $q_{\pi}(s, a)$ 를 계산할 때는 s 에서 강제로 a 를 선택합니다. 일단 액션을 선택한 이후에는 2가지 가치 함수 모두 정책 함수 π 를 따라서 에피소드가 끝날 때까지 진행합니다.

이렇게 하여 2가지 가치 함수 v 와 q 에 대해 살펴보았습니다. 이후에 이 강의에서 아무 명시 없이 **가치 함수**라고만 쓴다면 **상태 가치 함수 v** 를 가리키는 말이며, q 를 지칭할 때는 **액션 가치 함수**라고 명시하겠습니다. 또한 이후의 내용에서는 표기의 단순화를 위해 중요하지 않은 경우 종종 **아랫 첨자 π** 가 생략될 수 있습니다.

▶ Prediction과 Control

큰 흐름에서 강화 학습이라는 것을 문제와 솔루션으로 나누어서 본다면 이번 내용은 문제를 셋팅하는 단계에 해당한다고 볼 수 있습니다. 문제의 세팅이라 함은 주어진 상황이 있을 때 이를 MDP의 형태로 만들어서, MDP를 풀고자 하는 것입니다. 그래서 지금까지 MDP와 그 안의 여러 요소에 대해 설명했습니다. 그런데 여기서 MDP를 푼다는 것은 어떤 뜻일까요? MDP가 주어졌을 때, 즉 (S, A, P, R, γ) 가 주어졌을 때 우리가 관심 있어 하는 태스크는 크게 2가지로 *Prediction*과 *Control* 이 있습니다.

Prediction

π 가 주어졌을 때 각 상태의 벨류를 평가하는 문제.

Control

최적 정책 π^* 을 찾는 문제

MDP가 주어지면 우리의 목적은 크게 보아 위 2가지 중 하나입니다. 예컨대 알파고처럼 바둑이라는 상황을 MDP로 표현하여 해당 MDP에서 최적 정책 (optimal policy) π^* 을 찾거나(어떠한 상대를 만나도 다 이기는 정책), 임의의 정책 π 에 대해 각 상태의 벨류 $v_\pi(s)$ 를 구하고자 하는 것입니다. Prediction과 Control에 대해 조금 더 알기 쉽게 그리드 월드라는 예시를 통해 설명해 보겠습니다.

출발	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	종료

- ✓ 보상 : 스텝마다 -1
- ✓ 정책 : 4방향 랜덤 움직임

위 그림의 그리드 월드와 같은 상황을 생각해 보겠습니다. 출발에서 시작하여 종료까지 도착하면 한 에피소드가 끝나며 **스텝마다 -1의 보상을 받는 상황**입니다. 따라서 **누적 보상을 최대화 하고자 한다면 최단 경로를 지나 종료 상태에 도착**해야 합니다. 이때 에이전트가 선택할 수 있는 액션은 총 4개로 동, 서, 남, 북 방향으로 한 칸 움직이는 것입니다. (벽에 도달해서 벽 방향으로 움직이면 제자리에 머물게 됩니다.) 그리드 월드는 매우 간단한 MDP의 한 예시입니다.

먼저 그리드 월드에서의 Prediction 문제에 대해 생각해 보겠습니다. **Prediction 문제에서는 일단 정책 π 가 하나 주어져야 합니다.** π 는 단순히 4방향으로 랜덤하게 움직이는 정책이라고 해봅시다. 수식을 통해 표현하면 다음과 같습니다.

$for\ all\ s\ \in\ S$

$\pi(동|s) = 0.25,\quad \pi(서|s) = 0.25,\quad \pi(남|s) = 0.25,\quad \pi(북|s) = 0.25$

어느 상태에서나 동, 서, 남, 북으로 움직일 확률이 고르게 분포되어 있는 정책입니다. 이때 아무 상태나 하나 잡아 봅시다. **예컨대 s_{11} 의 벨류 $v_{\pi}(s_{11})$ 의 값은 몇이 될까요?** 이것이 바로 **Prediction 문제**입니다. 해당 상태의 벨류를 예측하는 것이 목적입니다.

여기서 “너무 쉽다, 그냥 아래로 한 칸 움직이면 -1의 보상을 받고 끝나니까 s_{11} 의 가치는 -1 아니겠어?” 라고 생각한다면 오산입니다.

왜냐하면 s_{11} 에서 출발하여 종료 상태까지 도달할 때 까지 4방향으로 랜덤하게 움직이므로 정말 다양한 경로가 가능하기 때문입니다. $\gamma = 1$ 이라고 한다면 다음과 같은 다양한 경로에 대해 다양한 리턴을 받을 수 있습니다.

출발	s_1	s_2	s_3
s_4	s_5	s_6	s_7
s_8	s_9	s_{10}	s_{11}
s_{12}	s_{13}	s_{14}	종료

$s_{11} \rightarrow$ 종료, 리턴 = -1
 $s_{11} \rightarrow s_{10} \rightarrow s_{14} \rightarrow$ 종료, 리턴 = -3
 $s_{11} \rightarrow s_7 \rightarrow s_6 \rightarrow s_{10} \rightarrow s_{11} \rightarrow$ 종료, 리턴 = -5

이 각각의 여정을 에피소드(Episode)라고 부릅니다. 이 모든 에피소드가 실제로 발생할 수 있습니다. 왜냐하면 에이전트는 마치 어린아이 처럼 4방향 무작위로 움직이기 때문입니다. 결국 에피소드마다 미래에 얻을 보상이 다르며, 또 그 에피소드가 발생할 확률도 다릅니다. 확률과 리턴의 곱을 모두 더해 줘야 상태 s_{11} 에서의 리턴의 기댓값을 구할 수 있게 되는 것이죠. 물론 가능한 경로가 무한히 많기 때문에 실제 가치 함수의 값을 그런 식으로 구하지는 않습니다. 게다가 만일 π 가 4방향 랜덤이 아니라 북쪽에 조금 더 치우친 확률 분포를 갖고 있는 정책이라면 $v_{\pi}(s_{11})$ 를 계산하는 것은 더 복잡해 집니다. 구체적으로 벨류 값을 구하는 방법은 차근차근 배워보겠습니다.

여기서 하고 싶은 주요 포인트는 그만큼 벨류를 구하는 문제가 간단하지는 않은 문제라는 것입니다.

CHAPTER 1 – 마르코프 결정 프로세스

이번에는 Control 문제를 살펴 보겠습니다. Control의 목적은 최적의 정책 π^* 을 찾는 것입니다. 최적의 정책이란 이 세상에 존재하는 모든 π 중에서 가장 기대 리턴이 큰 π 를 뜻합니다. 그리드 월드의 경우 π^* 는 비교적 간단합니다. 가장 빠른 경로를 향해서 나아가면 됩니다. 하지만 일반적인 MDP에서 π^* 를 찾는 것은 간단하지만은 않습니다. 이를 찾기 위해 강화 학습을 사용하는 것입니다.

또 최적 정책 π^* 를 따를 때의 가치 함수를 최적 가치 함수(optimal value function)라고 하며 v^* 라고 표기합니다. π^* 와 v^* 을 찾았다면 “이 MDP는 풀렸다”고 말할 수 있습니다. 결국 우리는 강화 학습을 이용해 실생활의 어떤 문제를 MDP 형태로 만들고, 그 MDP의 최적 정책과 최적 가치 함수를 찾아내며 MDP를 푸는 것이 목적입니다.

π^* 와 v^* 을 찾았다면 MDP는 풀렸다.

Prediction과 Control이라는 틀에서 강화 학습 문제를 바라보면 더 폭넓은 이해가 가능합니다.
이제 본격적으로 해결 방법에 대한 내용으로 넘어가 보겠습니다.

